**UNIVERSIDAD CARLOS III DE MADRID**

**ESCUELA POLITÉCNICA SUPERIOR**

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA


**PROYECTO FIN DE CARRERA**

Ingeniería Técnica Industrial: Electrónica Industrial

# APLICACIONES DEL OPERADOR SURF AL ANÁLISIS DE IMÁGENES

**AUTOR:** Rogelio Alarcia Noci

**TUTOR:** Arturo de la Escalera Hueso

Julio 2011

# ÍNDICE

# I. INTRODUCCIÓN

# I.INTRODUCCIÓN

# I INTRODUCCIÓN

## 1.1 Introducción a la Visión Artificial

La vista es uno de los sentidos más importantes para el ser humano. Mientras que para el oído se tiene alrededor de treinta mil terminaciones nerviosas, en la vista hay más de dos millones. Es el sentido que más información nos aporta sobre el entorno físico que nos rodea. Se estima que más de un 70% de los las tareas llevadas a cabo por el cerebro son empleadas en el análisis de la información visual.

Aristóteles en el Libro Primero de Metafísica dice *"Todos los hombres tienen naturalmente el deseo de saber. El placer que nos causa las percepciones de nuestros sentidos es una prueba de esta verdad. Nos agradan por sí mismas, independientemente de su utilidad, sobre todo las de la vista. En efecto, no sólo cuando tenemos intención de obrar, sino hasta cuando ningún objeto práctico nos proponemos, preferimos, por decirlo así, el conocimiento visible a todos los demás conocimientos que nos dan los demás sentidos. Y la razón es que la vista, mejor que los otros sentidos, nos da a conocer los objetos, y nos descubre entre ellos gran número de diferencias".*

La visión humana es el sentido más desarrollado y también el más desconocido debido a su gran complejidad. Es una actividad inconsciente y es difícil saber cómo se produce.

En la actualidad sigue sin haber una teoría de la visión. Se desconocen los procesos del cerebro para extraer la información de las imágenes que captan los ojos. El cerebro es capaz, de manera inconsciente, de determinar la distancia a los objetos, de reconocerlos en diferentes posiciones, rotados, parcialmente ocultos o degradados.

La visión artificial pretende capturar la información visual del entorno físico para extraer características relevantes visuales, utilizando procedimientos automáticos.

Es fácil ver la analogía entre la visión humana y la visión por computador. En ambos casos existe un elemento sensor, el ojo y la cámara. Y un elemento que procesa la información, el cerebro en el primer caso, y el computador en el segundo.

En la década de los 50, cuando se establecieron los objetivos de la Inteligencia Artificial y se empezaron a desarrollar los primeros trabajos en el campo de la visión por computador. Se esperaba que esta tecnología se desarrollase rápidamente. "¿Si

es fácil ver para un humano por qué no lo va a ser para una máquina?" Con el paso del tiempo aquel entusiasmo se fue desvaneciendo. Aunque se desarrollaron algoritmos que son utilizados actualmente, los computadores de la época no estaban a la altura.

A partir de los ochenta, con el desarrollo de nuevos computadores más rápidos, con mayor capacidad de cálculo y más económicos, junto con un enfoque más realista de la visión por computador, este tema volvió a ser objeto de desarrollo.

La visión por computador se relaciona con las siguientes disciplinas:

- **Fotografía y óptica.** Crear el ambiente de iluminación adecuado en la adquisición de imágenes Es necesario hacer una correcta selección de la cámara, la óptica, y los medios de iluminación, según la aplicación que se pretenda dar.
- **Procesamiento de las imágenes.** La finalidad es transformar la imagen capturada en otra que pueda tener mayor relevancia.
- **Generación de gráficos por computador.** Es el proceso inverso al análisis de imágenes. Se extrae la imagen a partir de la descripción.
- **Reconocimiento de patrones.** Técnica dedicada a la búsqueda y clasificación de objetos a partir de unas características.

Con el paso del tiempo y el desarrollo de hardware y software específico, la visión por computador se empieza a introducir en muchos tipos de aplicaciones, que son variadas y siguen aumentando en cantidad:

- **Robótica:** Guiado de robots industriales, navegación de robots móviles.
- **Control de calidad:** Inspección de productos, identificación de piezas, etiquetado…
- **Seguridad:** Detección, vigilancia.
- **Control de tráfico:** Identificación de matrícula, reconocimiento de señales y líneas de carretera.
- **Militares:** Seguimiento de objetivo, vigilancia por satélite.
- **Biomedicina:** Análisis de imágenes microscópicas, resonancias magnéticas, tomografías, análisis de imágenes tomadas por rayos X, ultrasonidos…
- **Identificación:** Caras, huellas dactilares…

Aunque todavía no es posible implementar artificialmente las tareas del cerebro en cuanto a la percepción, y se está muy lejos de conseguir algo equivalente a la visión

humana, la visión por computador es muy eficaz en tareas visuales repetitivas y alienantes para el humano, y no está sometida al cansancio ni al estado de ánimo. A parte de ser mucho más eficaz en tareas como contar elementos en una imagen, o determinar la trayectoria de un robot.

## 1.2 Reconocimiento de objetos

El desarrollo de una aplicación basada en visión por computador se puede dividir en dos partes. La primera, visión de bajo nivel, donde se pretende procesar la imagen obtenida para obtener las características útiles para resolver el problema. La segunda es la visión de alto nivel o análisis de imágenes, donde se usan las características obtenidas en la primea parte para llegar a descripciones más abstractas, como el análisis de formas y el reconocimiento y localización de objetos.

El reconocimiento de objetos en visión por computador es la tarea de encontrar un objeto dado en una imagen. El ser humano puede reconocer objetos en imágenes con poco esfuerzo, aunque éste cambie en orientación, iluminación, distancia o esté parcialmente oculto. En un sistema de visión por computador el objetivo es el mismo.

Con este objeto se desarrollan clasificadores que a partir de una imagen buscan puntos característicos y crean descriptores asociados, de forma que se puedan emplear para localizar esos mismos puntos en otra imagen donde aparezca el mismo objeto, aunque haya variado su posición, orientación, iluminación...

## 1.3 Objetivo

El objetivo fijado para este proyecto es desarrollar un programa en lenguaje C++, basado en el detector y descriptor SURF (Speeded-Up Robust Features).

Este programa se encargará de buscar y señalar la posición un objeto en cada imagen de un conjunto de tamaño indeterminado, donde este objeto puede aparecer visto desde distintos ángulos, diferente distancia o parcialmente oculto.

Para ello el algoritmo necesitará una imagen “modelo” de la que extraerá los puntos de interés junto con sus descriptores, para luego tratar de localizar esos mismos puntos en las demás imágenes.

# II. SURF

## II.SURF

# II SURF

## 2.1 SURF (Speeded Up Robust Features)

El algoritmo SURF fue desarrollado por Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool y presentado por primera vez el año 2006.

Las características principales que los creadores le atribuyeron a este detector y descriptor, son el permanecer invariante frente a rotaciones y cambios de escala, y una alta velocidad de cálculo.

En la mayoría de las aplicaciones de visión por computador, la tarea principal es encontrar correspondencias entre puntos de dos imágenes. Esta búsqueda de correspondencias se puede dividir en tres etapas:

- Búsqueda de “puntos de interés”, tales como bordes, manchas, esquinas. La propiedad más importante de los descriptores es la repetitividad. La repetitividad expresa la fiabilidad del detector para encontrar el mismo punto de interés en diferentes condiciones.
- El entorno de un punto de interés se representa por un vector de características. Esto es el “descriptor” del punto de interés, y debe ser distintivo, y robusto frente a ruido, desplazamientos y deformaciones.
- El último paso es la localización de los descriptores en diferentes imágenes. Esta correspondencia se basa en la distancia entre vectores (como la distancia Mahalanobis o la Euclidea).

A continuación se describe el funcionamiento de estos pasos en SURF.

## 2.2 Detección de puntos de interés

Hay varios detectores que se utilizan para localizar los puntos de interés. El detector de esquinas de Harris, propuesto en 1988, posiblemente ha sido de los más usados. Sin embargo se ve afectado por los cambios de escala. Más tarde, Lindeberg introdujo el concepto de selección de escala automática. De esta forma es posible detectar puntos de interés en una imagen, independientemente de sus características de

escala. Lindeberg experimentó con las matrices Hessianas y el Laplaciano (que equivale a la traza de la Hessiana). Mikolajczk y Schmid refinaron el método, creando unos detectores robustos e invariantes a la escala con alta repetitividad, a los que llamaron Harris-Laplace y Hessian-Laplace. Usaron el detector Harris en el primero, y el determinante de la matriz Hessiana en el segundo, para la localización, y el Laplaciano en ambos para seleccionar la escala. Lowe, centrándose en la velocidad, en su detector SIFT propuso una aproximación de Laplaciano de Gaussianas (LoG) usando un filtro de Diferencia de Gaussianas (DoG).

Existen otros tipos de detectores que no se ven afectados por las variaciones de escala, y a partir de estudios comparativos entre ellos, se puede concluir que los detectores Hessianos son más estables y repetitivos que los basados en Harris. Incluso se puede decir que usar el determinante de la matriz Hessiana es más ventajoso que el uso de su traza (el Laplaciano), perdiendo menos tiempo en estructuras alargadas y mal localizadas.

SURF, en la localización de puntos de interés, usa una aproximación muy básica de la matriz Hessiana, y depende de imágenes integrales (Viola y Jones) para conseguir una reducción drástica del coste computacional.

### *2.2.1 Imagen integral*

La imagen integral de Viola y Jones es un método para realizar de forma rápida el cálculo de nivel medio de intensidad de las regiones de píxeles en una imagen o matriz. El valor $I_\Sigma$ de la imagen integral en el punto $X=(x,y)^T$ representa la suma de todos los píxeles de la imagen original $I$ de la región rectangular que queda por encima y a la izquierda de X.

$$I_\Sigma(X) = \sum_{i=0}^{i\leq x} \sum_{j=0}^{j\leq y} I(i,j)$$

Si cada elemento de la imagen integral tiene la suma de todos los píxeles localizados en la región superior izquierda de la imagen original, sólo son necesarias cuatro búsquedas para calcular la suma de intensidades dentro de una zona rectangular de cualquier tamaño.

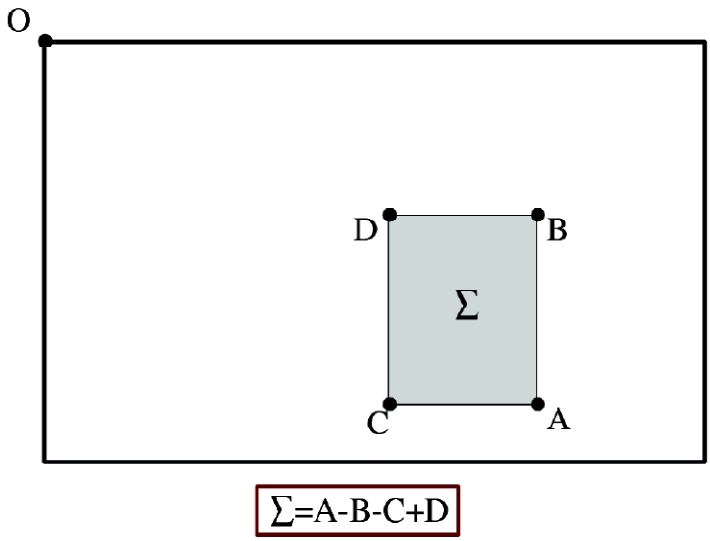



Figura 2.1 Empleo de imágenes integrales,

### *2.2.2 Matriz Hessiana en la búsqueda de puntos de interés*

Los creadores de SURF basan este detector en la matriz Hessiana por su buen comportamiento en cuanto a la relación precisión y tiempo de cálculo. En concreto, el algoritmo detecta “zonas-mancha” en lugares donde el determinante es máximo.

A diferencia del detector Hessiano-Laplaciano de Mikolajczyk y Schmid, se usa también el determinante de la matriz Hessiana para la selección de escala, tal como hizo Lindeberg.

Dado un punto X=(x,y) en una imagen $I$, la matriz Hessiana H(X,$\sigma$) en el punto X y a escala $\sigma$ se define de la siguiente manera:

$$\mathrm{H}(\mathrm{X},\sigma) = \begin{bmatrix} \mathrm{L_{xx}}(\mathrm{X},\sigma) & \mathrm{L_{xy}}(\mathrm{X},\sigma) \\ \mathrm{L_{xy}}(\mathrm{X},\sigma) & \mathrm{L_{yy}}(\mathrm{X},\sigma) \end{bmatrix}$$

Donde $\mathrm{L_{xx}}(\mathrm{X},\sigma)$ es la convolución de la derivada de segundo orden de la Gaussiana $\frac{\delta^2}{\delta \mathrm{x}^2}\mathrm{g}(\sigma)$ con la imagen $I$ en X. Para $\mathrm{L_{xy}}(\mathrm{X},\sigma)$ y $\mathrm{L_{yy}}(\mathrm{X},\sigma)$ el proceso es similar.

En el análisis de la escala, el uso de Gaussianas es óptimo, pero para su aplicación práctica es necesario discretizarlas y cuantificarlas (figura 2.3).

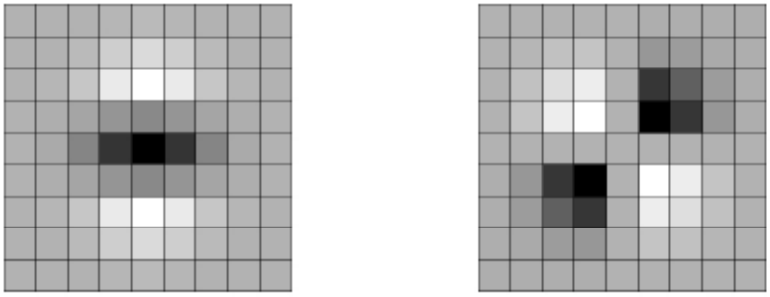


Figura 2.2: La primera imagen corresponde a la derivada de segundo orden de la Gaussiana (discretizada y cuantificada) en y (Lyy). La segunda imagen corresponde a Lxy.

Esto lleva a una pérdida de repetitividad al rotar la imagen en algunos múltiplos de $\pi/4$. Y esta repetitividad alcanza valores máximos en los múltiplos de $\pi/2$ debido a la forma cuadrada del filtro. Esta situación se da en todos los detectores basados en matrices Hessianas. A pesar de esto, el comportamiento del detector sigue siendo bueno, y se puede asumir esta pequeña pérdida de repetitividad frente a la ventaja que supone la alta velocidad de cálculo que proporciona la discretización y cuantificación. En la figura 2.4 se muestra el comportamiento de la repetitividad de dos detectores basados en matrices Hessianas frente a la rotación.

Basándose en que ningún filtro es ideal, y en el buen comportamiento observado del Laplaciano de las Gaussianas (LoG) usado por el algoritmo SIFT años antes, en SURF se decide simplificar aun más utilizando unos filtros-caja (Figura 2.3). Este filtro es una aproximación de la derivada de segundo orden de la Gaussiana.

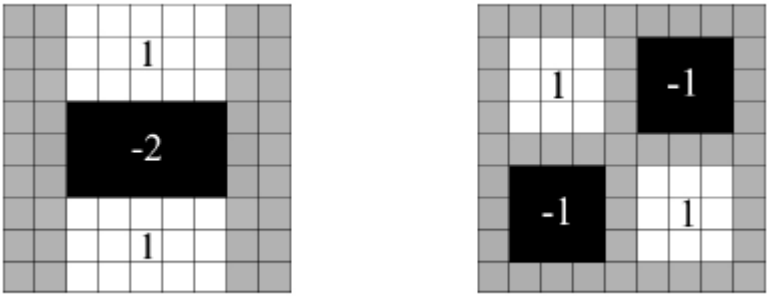



Figura 2.3: Aproximación de la derivada de segundo orden de la Gaussiana en y (Dyy). La segunda imagen corresponde a Dxy. Las regiones en gris tienen valor 0. Estas aproximaciones se extraen de las imágenes de la figura 2.2.

Esta aproximación a las derivadas de segundo orden de la Gaussiana puede ser evaluada con muy poco coste computacional usando las imágenes integrales. Como se ha visto en el punto anterior, el tamaño del filtro no afectaría al tiempo de cálculo.

En los resultados de la figura 2.4 se comparan los dos casos, con el uso de la discretización y cuantificación de las derivadas de segundo orden de la Gaussiana y con el método de la aproximación, y los resultados además de ser similares, son incluso mejores en el segundo caso.

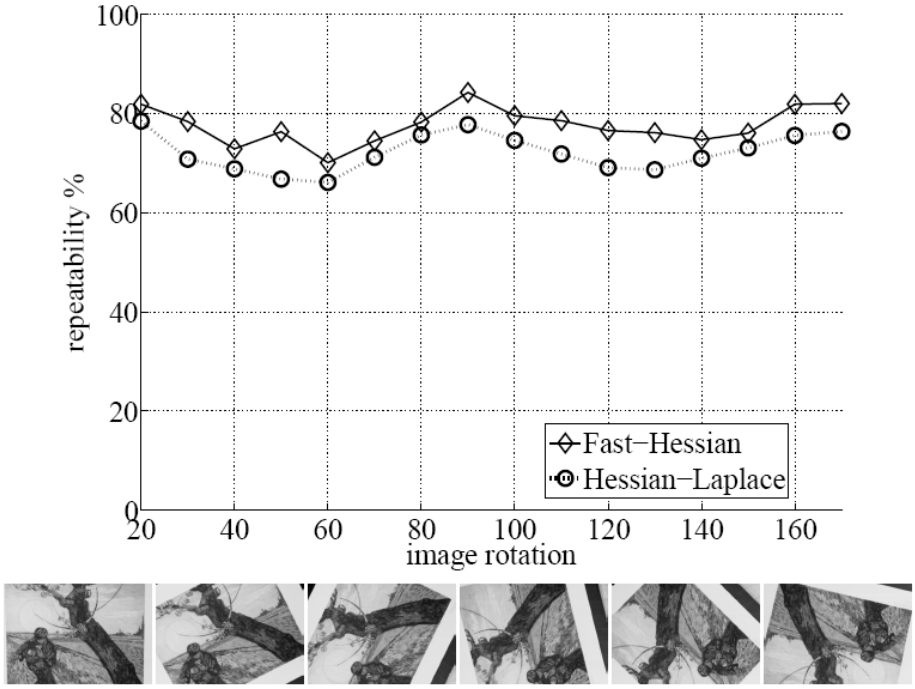



Figura 2.4: Comportamiento de la repetitividad durante la rotación de una imagen. Los puntos mínimos se corresponde con múltiplos de $\pi/_4$.

Estas aproximaciones de $L_{xx}(X,\sigma)$, $L_{yy}(X,\sigma)$ y $L_{xy}(X,\sigma)$ se llamaran $D_{xx}$, $D_{yy}$ y $D_{xy}$. Para conservar la eficiencia computacional, se asignan valores simples al peso de las regiones rectangulares (figura 2.3).

De esta forma el cálculo del determinante de la matriz Hessiana queda así:

$$\det(H_{aprox}) = D_{xx}D_{yy} - (wD_{xy})^2$$

Donde w es el peso relativo de la respuesta del filtro y se usa para regular la expresión del determinante.

El determinante aproximado de la matriz Hessiana se utiliza para la localización de puntos y para determinar la escala.

Para conservar el equilibrio entre la solución sin aproximación y la aproximada, por ejemplo si se usa un filtro 9x9 (el más pequeño) para una Gaussiana con σ=1,2, se obtiene que w=0,9.

$$w = \frac{|L_{xy}(1,2)|_F |D_{yy}(9)|_F}{|L_{yy}(1,2)|_F |D_{xy}(9)|_F} \backsimeq 0,9$$

Según los autores de SURF, un uso teóricamente correcto, supone que el valor w sea dependiente de la escala, pero en los experimentos realizados no ha supuesto ninguna alteración destacable el tratar este valor como una constante.

Además, la respuesta del filtro se normaliza con respecto de su tamaño, siendo constante la norma de Frobenius para cualquier tamaño de filtro.

### *2.2.3 Representación espacial de la escala*

El espacio de escalas se suele implementar como una pirámide. Las imágenes se suavizan repetidamente con una Gaussiana y después se submuestrean para alcanzar un nivel más alto en la pirámide. Para el detector SIFT, Lowe extrae las capas de la pirámide para conseguir mediante DoG (Diferencia de Gaussianas) las imágenes donde es posible localizar bordes y zonas-mancha. En SURF, gracias al uso de filtros-caja e imágenes integrales, no es necesario aplicar iterativamente el mismo filtro a la salida de una capa previamente filtrada. En lugar de eso, se usa el filtro-caja del tamaño conveniente, a la misma velocidad (independiente del tamaño) partiendo directamente de la imagen original, incluso siendo posible el filtrado en paralelo en los distintos tamaños. De esta forma, el espacio de escala se analiza aumentando el tamaño del filtro (escala ascendente), en lugar de reducir iterativamente el tamaño de la imagen (figura 2.5) como se hacía en SIFT.

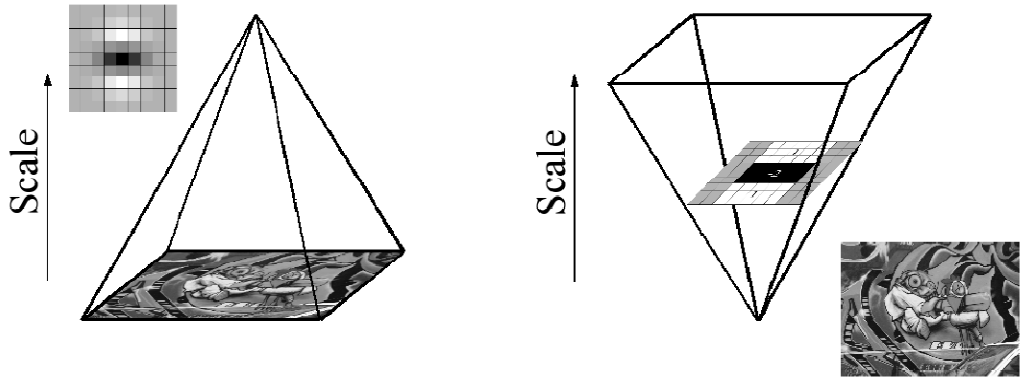



Figura 2.5: Reducción iterativa de SIFT (izquierda). El uso de imágenes integrales en SURF permite aumentar la escala con un coste computacional constante (derecha).

Al resultado del filtro 9x9 se le considera la primera capa de escala. A esta escala se la denomina s=1,2 (aproximada de la derivada de la Gaussiana con $\sigma$=1,2). Las siguientes capas se obtienen usando filtros mayores, teniendo en cuenta la naturaleza discreta de las imágenes integrales y la estructura específica de los filtros-caja.

Este espacio de escala se divide en octavas. Una octava representa una serie de mapas resultado de un filtrado, obtenida a partir de una misma imagen de entrada con un filtro de tamaño creciente. Cada octava se divide en un número constante de niveles. En la primera octava, la que parte del filtro 9x9 para pasar a un nivel siguiente se deben sumar 6 píxeles. En las siguientes octavas el número de pixeles se va doblando del siguiente modo:

- Primera octava →9x9→+6→15x15→+6→21x21→+6→27x27
- Segunda octava →15x15→+12→27x27→+12→39x39→+12→51x51
- Tercera octava →27x27→+24→51x51→+24→75x75→+24→99x99

La construcción del espacio de escala empieza con el filtro 9x9, que localiza las zonas-mancha de la imagen en la escala más pequeña. Después se usan los filtros 15x15, 21x21 y 27x27. El proceso es el mismo con otras octavas. En la figura 2.6 sólo se muestran las 3 primeras octavas, es posible aplicar más, pero el número de puntos de interés encontrados disminuye rápidamente con octavas mayores.

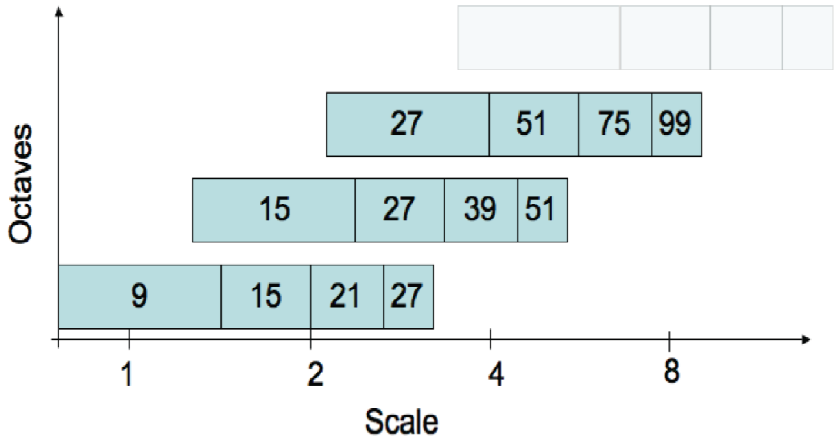



Figura 2.6: Representación gráfica de los tamaños de filtro para tres octavas diferentes. El eje x (logarítmico) representa la escala.

### *2.2.4 Localización de los Puntos de Interés*

Para localizar los puntos de interés en la imagen original y en las escaladas, se procede a la eliminación de los puntos que no sean máximos en la región vecina 3x3x3. El máximo determinante de la matriz Hessiana se interpola en la escala y el espacio de la imagen. En la siguiente figura se muestra un ejemplo de imagen con puntos de interés localizados mediante un detector Hessiano.

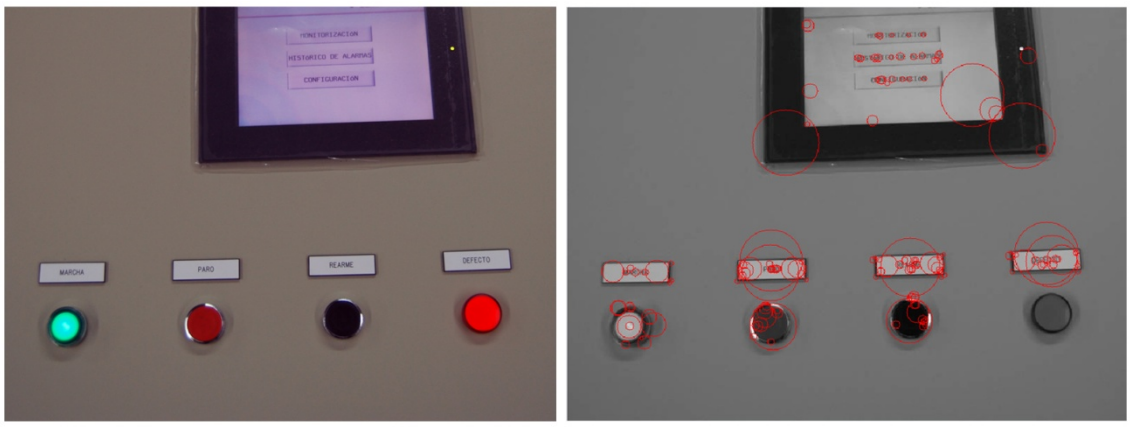



Figura 2.7: Imagen derecha muestra puntos de interés detectados por SURF. La imagen izquierda es la original

## 2.3 Descripción del Punto de Interés

Una vez localizados los puntos de interés, los siguientes pasos son la asignación de una orientación, basada en la información de una zona circular alrededor del punto de interés Después se construye una región cuadrada alineada con dicha orientación y se extrae de ella el descriptor SURF. Este descriptor utiliza como información la distribución de la intensidad alrededor del punto de interés.

### *2.3.1 Asignación de la orientación*

Con la intención de que el detector sea invariante a la rotación, se asigna una orientación a los puntos de interés que es reproducible en otras imágenes. El primer paso es calcular las funciones de Haar en las direcciones x e y dentro de una región circular de radio 6s, donde s es la escala a la que el punto de interés se ha detectado. El tamaño de las funciones (ondulas de Haar) también se hace de forma dependiente a la escala, ajustando la longitud a 4s. Aquí es posible emplear de nuevo las imágenes integrales para realizar un filtrado rápido. Sólo son necesarias 6 operaciones para obtener las respuestas en x o y a cualquier escala.

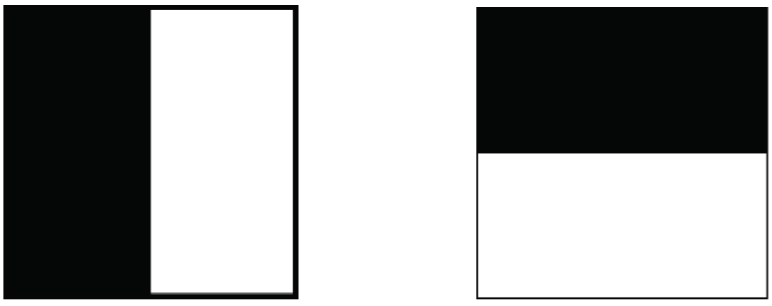


Figura 2.8: Respuesta a los filtros de las wavelets de Haar en la dirección x (izquierda) e y (derecha). A las partes oscuras se le asigna el peso -1 y a las claras +1.

Con los resultados de la wavelet calculados y ponderados con una Gaussiana ($\sigma = 2s$) centrada en el punto de interés, se representan de estos resultados como puntos en un espacio de dos dimensiones donde las respuestas horizontales se distribuyen a lo largo de la abscisa y las verticales a lo largo de la ordenada. La orientación dominante se calcula sumando todos los puntos resultantes dentro de una ventana deslizante de tamaño $\pi/3$ (figura 2.9). Al sumar los resultados horizontales y los verticales se obtiene el vector de la orientación local. El vector más largo de todas las ventanas es el que define la orientación del punto de interés. El tamaño de la ventana es un parámetro que debe ser elegido cuidadosamente. Tamaños pequeños o grandes llevan a malos resultados.

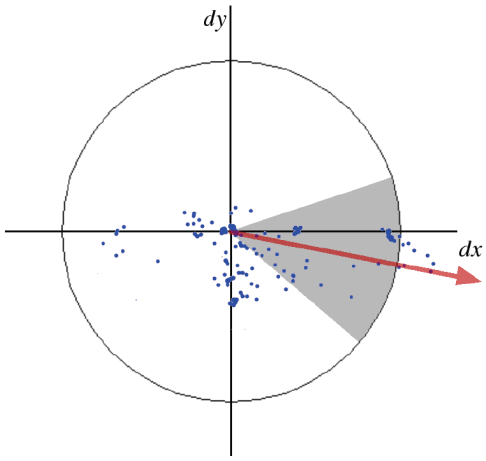



Figura 2.9: Asignación de orientación. La ventana deslizante definida por un arco de 60º detecta la orientación dominante de los resultados de la Gaussiana de la wavelet de Haar para cada punto dentro de la región circular creada alrededor del punto de interés.

### *2.3.2. Descriptor*

El descriptor se basa en la suma de los resultados de las wavelets de Haar. El primer paso para la construcción de este descriptor consiste en la construcción de una región cuadrada centrada en el punto de interés y orientada según el resultado obtenido en el aparatado anterior. El tamaño asignado a cada ventana es 20s.

Figura 2.10: Ejemplo de imagen que muestra la orientación y el tamaño de las ventanas de los descriptores a diferentes escalas.

Una vez se tiene la región cuadrada de tamaño 20s, se divide esa región en 4x4 sub-regiones. Y a su vez estas se dividen en otros elementos cuadrados de 2x2, obteniendo al final una rejilla de 64 elementos. Para cada sub-región se calculan los resultados de las wavelets de Haar para x e y. A la respuesta a la función de Haar en la dirección horizontal se le da el nombre $d_x$, y en la dirección vertical $d_y$. En este caso "horizontal" y "vertical" se definen en relación a la orientación del punto de interés seleccionado (figura 2.11).

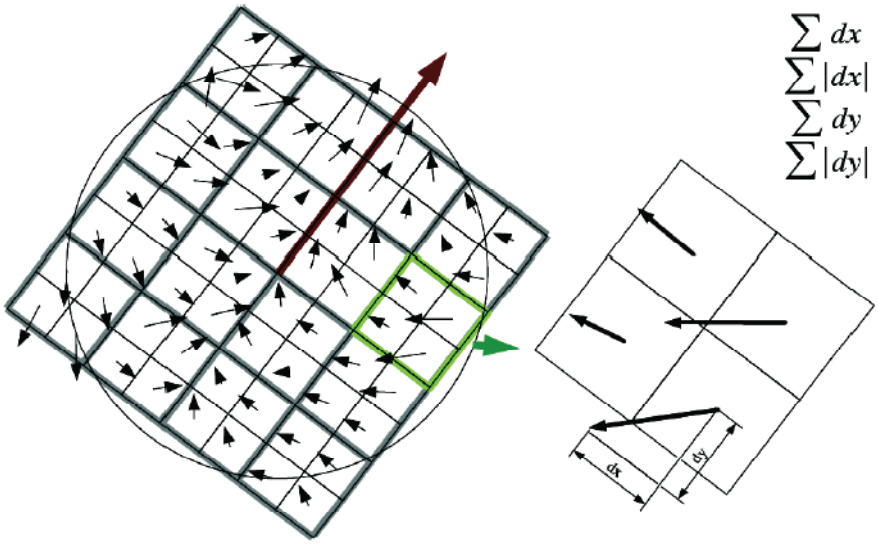



Figura 2.11: Para construir el descriptor, se planta una rejilla orientada de 4x4 sub-regiones cuadradas (izquierda). A cada cuadro se le aplica una subdivisión 2x2 (derecha), quedando un total 64 elementos. Para cada cuadrado se calcula la respuesta de la wavelet.

Con los resultados de las wavelets obtenidos $d_x$y $d_y$ se hace la suma en cada sub-región, y a partir de esta información se obtiene el vector de 4 dimensiones:

$$v = (\Sigma d_x, \Sigma d_y, \Sigma|d_x|, \Sigma|d_y|)$$

Concatenando todas las sub-regiones 4x4, el resultado es un vector descriptor de dimensión 64.

## 2.4 Emparejamiento/Matching

La última etapa del detector consiste en el emparejamiento de los puntos localizados en la imagen inicial con los de una segunda. Se comparan las características extraídas y se establece la relación. Durante esta fase los autores proponen tener en cuenta el uso del signo del Laplaciano (la traza de la matriz Hessiana). Este signo distingue las manchas oscuras sobre fondo claro de las claras sobre fondo oscuro. Este dato no supone aumento del coste computacional, puesto que ya se ha hecho durante la etapa de detección, y es útil para descartar del proceso de comparación los puntos con Laplaciano de distinto signo.

### *2.4.1 Métodos de emparejamiento usados (FLANN y naive)*

En este proyecto se incluyen dos métodos de “matching” válidos. Cualquiera de los dos métodos, aparte de lo comentado anteriormente, se basa en la búsqueda del vecino más próximo. El problema del “vecino más próximo” se puede definir de la siguiente manera: Dado un conjunto de puntos $\boldsymbol{P} = \{\boldsymbol{p_1}, \dots, \boldsymbol{p_n}\}$ en un espacio euclídeo $X$, estos puntos deben ser preprocesados de forma que dado un nuevo punto $\boldsymbol{q} \in \boldsymbol{X}$, sea posible encontrar eficientemente los puntos de $P$ más próximos a $\boldsymbol{q}$.

Los dos métodos usados en el programa se conocen como FLANN y naive.

FLANN (Fast Library for Approximate Nearest Neighbors), en realidad es una biblioteca, desarrollada y escrita en C, para la búsqueda del vecino más próximo en espacios de muchas dimensiones. Contiene varios algoritmos que pueden ser utilizados para este fin (Randomized KD-tree, Hierarchical K-means tree).

Con el método basado en FLANN, el programa detecta los puntos de interés y sus características, después con esta información usa cuatro Randomized KD-Trees (configurado por defecto) y busca los dos vecinos más próximos. El punto emparejado sólo es válido si la distancia al segundo vecino más cercano es el doble de la distancia al primero.

El otro método, “naive”, es un sistema primitivo de emparejamiento. En este caso se hace una búsqueda lineal de cada punto de interés de la imagen del objeto en la imagen que contiene a éste. Se hace una búsqueda de los dos vecinos más cercanos y se comprueba que el primero de los dos es dominante antes de dar el resultado por válido.

La distancia considerada entre puntos es la distancia euclídea.

### *2.4.2 Homografía*

Una vez aplicado el emparejador, el programa aplica un sistema que permite hacer una restricción geométrica en la imagen de destino, para delimitar el objeto encontrado, e incluso es capaz de mostrar el movimiento relativo entre ambas imágenes. Usando los resultados obtenidos de cualquiera de los métodos de

emparejamiento empleados, es posible obtener la matriz de transformación de perspectiva. Las cuatro esquinas de la imagen objeto se localizarán en la imagen de destino, delimitando el objeto con un polígono de cuatro lados.

El problema de la homografía se puede plantear así: Dado un conjunto de puntos en correspondencia $\mathbf{x_i} \leftrightarrow \mathbf{x'_i}$ entre dos imágenes, la matriz de homografía $\mathbf{H}$ es la matriz no singular 3x3 tal que $\mathbf{x'_i} = \mathbf{Hx_i}$ para todo i.

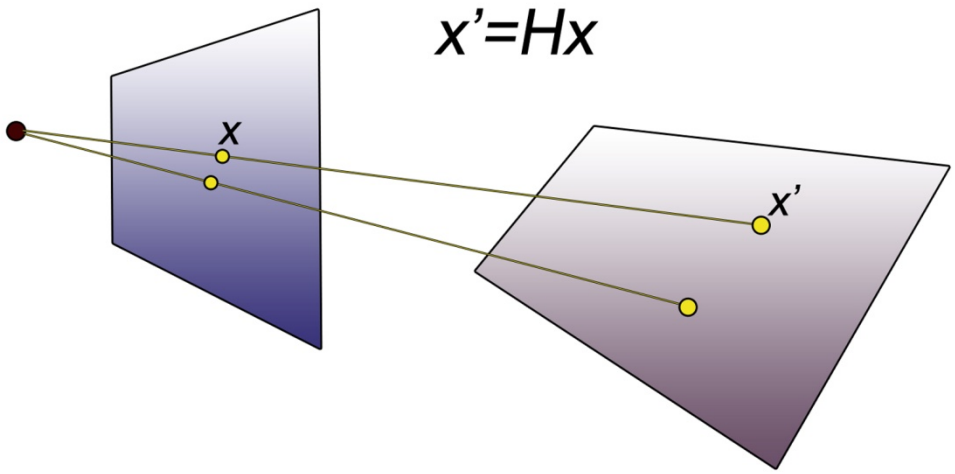



Figura 2.12: Paso del punto x a x' por homografía.

El programa calcula esta matriz a partir de los puntos emparejados conseguidos.

Es habitual que dentro del conjunto de emparejamientos, haya una cantidad de resultados falsos. Si se resuelve el sistema sin eliminarlos, la solución obtenida no es aceptable. Para que la matriz de homografía sea válida es necesario eliminar los emparejamientos espurios. Para conseguir un emparejamiento robusto existen varias técnicas que permiten eliminar los resultados no deseados. En este proyecto permite la configuración del programa para utilizar el método probabilístico de emparejamiento robusto RANSAC o el LMEDS.

### *2.4.3 RANSAC*

El método RANSAC (RANdom SAmple Consensus) obtiene cómo resultado la solución más votada de entre las calculadas a partir de conjuntos obtenidos aleatoriamente.

La idea básica de este método es la siguiente:

1) Se elijen dos puntos al azar del conjunto de la “nube de datos”. Estos dos puntos definen una recta.
2) El soporte a esta recta se mide por el número de puntos cuya distancia normal a dicha recta está dentro de un umbral seleccionado previamente. Los puntos dentro del umbral se denominan puntos de consenso.
3) Esta selección aleatoria se repite un número de veces y la recta con mayor soporte se considera como línea robusta.

Se utiliza la idea de que una línea, en la que uno de los puntos pertenece a un falso emparejamiento, no tendrá mucho soporte.

En el caso de una homografía plana el número mínimo de puntos para ajustar el modelo es de cuatro puntos en lugar de dos, que era el caso anterior.

El proceso es el siguiente, una vez encontrados los puntos de interés y calculados los posibles emparejamientos se seleccionan muestras aleatorias de 4 correspondencias y calcula la matriz de homografía, después se calcula el número de puntos que entran dentro del umbral. Esto se hace para un número de muestras N, donde N tiene un tamaño lo suficientemente alto como para garantizar con una alta probabilidad, que al menos uno de los subconjuntos no contiene falsos emparejamientos. Se elije como matriz de homografía final a la que tiene un mayor número de puntos dentro del umbral.

### *2.4.4 LMEDS*

LMEDS (Least MEDian of Squares) es el método de la mínima mediana de los cuadrados de los residuos, donde el residuo es la diferencia entre el valor estimado y el real.

**II.SURF**

Es un sistema similar al RANSAC. El proceso es paralelo al del método anterior salvo cuando llega el momento de la elección del resultado. En este caso en lugar de elegir el resultado más “votado” de los observados dentro de un grupo, el criterio cambia y se calcula la mediana del cuadrado de los residuos. Se toma como mejor estimación aquella cuyo valor mediano de residuos alcance el valor mínimo.

La ventaja de este método es que no es necesario encontrar un valor óptimo de tolerancia.

# III. RESULTADOS

## III.RESULTADOS

# III RESULTADOS

## 3.1 Descripción del funcionamiento del programa

Para el estudio experimental de los resultados de este algoritmo, se ha creado un software basado en una implementación previa de SURF realizada por el autor Liu-Liu.

El código del programa se ha desarrollado en C. La función de este programa es buscar un objeto en otra imagen que puede contener a este objeto. Se le da a la aplicación una imagen que identifica y utiliza como objeto a buscar. En otra carpeta se almacenan las imágenes donde se pretende encontrar a este objeto.

En los primeros pasos a aplicación crea una lista de las imágenes disponibles, después busca los puntos de interés en la imagen/objeto con SURF. En la pantalla de mensajes se informa de la cantidad de descriptores encontrados. A partir de ahí, durante su ejecución, el programa aplica el algoritmo SURF con otro algoritmo de emparejamiento a cada imagen, una por una, e informa del resultado al final de cada comparación mediante mensajes de texto, donde indica si hay resultado positivo o no, el número de descriptores y el tiempo de cálculo. Al mismo tiempo muestra cada resultado en una ventana, la primera ventana es para la imagen-objeto con sus descriptores, las siguientes ventanas contienen la imagen-objeto sobre la imagen de búsqueda junto con las líneas de correspondencia entre los puntos emparejados. En caso de que el resultado sea positivo en la imagen de búsqueda se señala el objeto encontrado dentro de un cuadrilátero.

Aunque es modificable, por defecto el programa busca la imagen objeto en la carpeta donde se encuentra el ejecutable. La imagen debe estar renombrada como “objeto.jpg”. Si el formato de la imagen es distinto a “jpg” se deberá convertir a este, o renombrarlo dentro del código. Las imágenes de destino también pueden guardarse en cualquier carpeta, pero por defecto, están en "C:/PFC/imagenes test/". No es necesario renombrarlas.


Figura 3.1: Ejemplo de resultado de la ejecución del programa en la búsqueda de un objeto en una imagen. La ventana de la izquierda muestra los puntos de interés localizados. La ventana de la derecha muestra la correspondencia de los puntos entre ambas imágenes y enmarca la solución.

## 3.2 Comprobación de funcionamiento

Para hacer una comprobación rápida del comportamiento del programa se propone la búsqueda de varios objetos en un entorno.

Se plantea la búsqueda de 3 objetos (figura 3.2) en 17 imágenes. Algunas de estas imágenes no contienen al objeto buscado y en otras es difícilmente visible debido a reflejos, o al ángulo y distancia, pero es suficiente para una observación inicial.


Figura 3.2: Objetos a buscar en varias imágenes de un entorno.

Se prepara una primera prueba empleando el método FLANN para el emparejamiento, y el método RANSAC para la eliminar falsas coincidencias, con un valor de umbral igual a 5.

Tras analizar la imagen objeto (figura 3.2), el programa va presentando las ventanas de resultados (figura 3.3) hasta completar la búsqueda en todas las imágenes propuestas, al mismo tiempo que muestra el informe de resultados en la pantalla de texto (figura 3.4).


Figura 3,2: Extracción de puntos característicos de una de las imágenes objeto.

## III.RESULTADOS

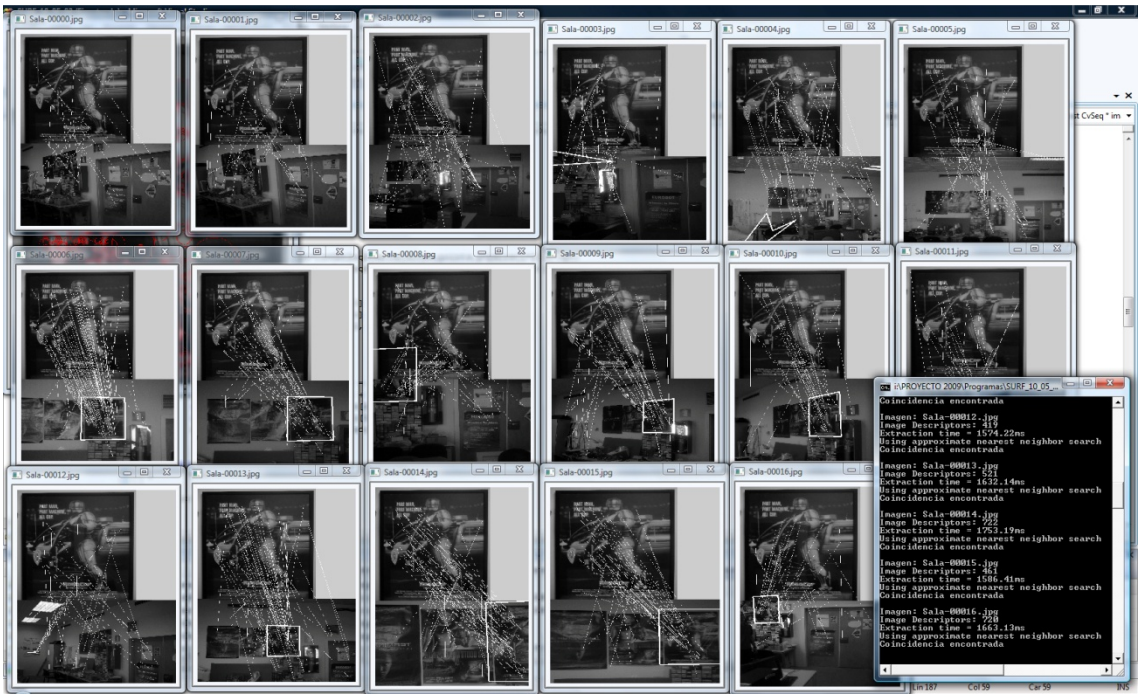



Figura 3.3: Muestra de resultados inicial con búsqueda configurada con FLANN y RANSAC umbral 5.

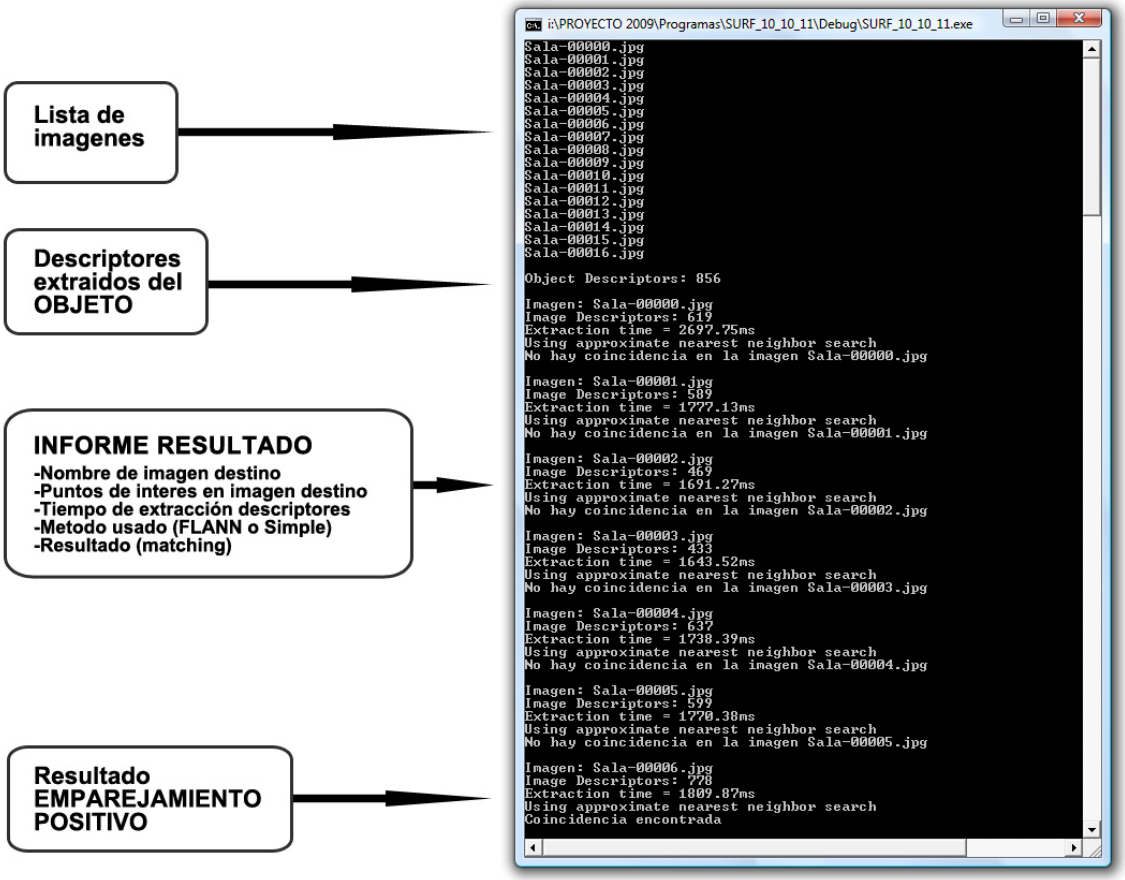



Figura 3.4: Ejemplo de ventana de texto.

## III.RESULTADOS

Al observar los resultados con la búsqueda de las 3 imágenes se deduce que cuando el enfoque es bueno, y la cantidad de brillos no es elevada en el objeto buscado el programa no tiene problemas para encontrarlo. Estas dos características parecen ser más perjudiciales, si no son buenas, incluso que cubrir parciamente el objeto o el efecto provocado por las rotaciones.

Cuando las condiciones no son tan buenas, o incluso si el objeto no está en la imagen, un gran número de veces el programa informa que encuentra coincidencias que no son aceptables.

Cambiando parámetros de búsqueda se determina rápidamente que es fácil mejorar la calidad de los resultados. El programa puede encontrar el objeto buscado en más imágenes y eliminar falsos resultados positivos, pero no se llega a eliminar totalmente los resultados espurios (figura 3.5).

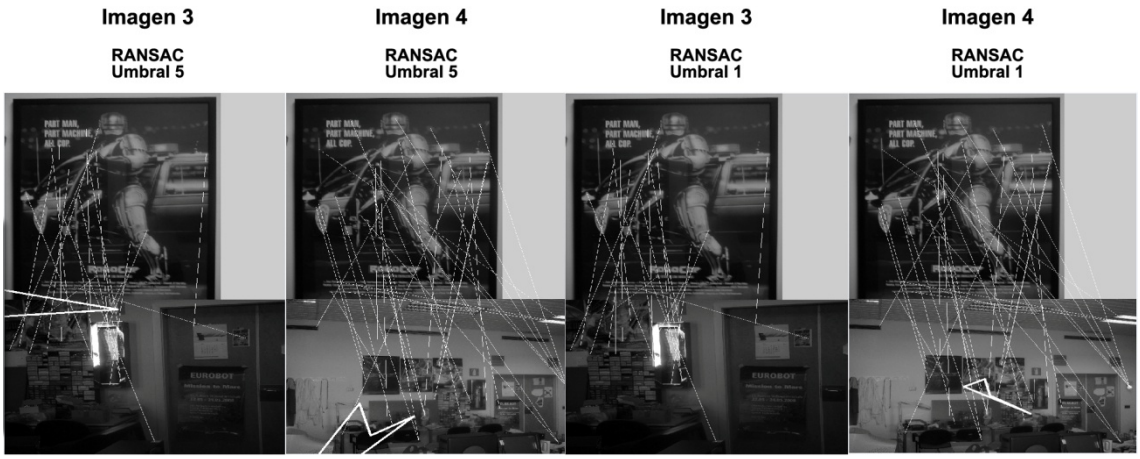



Figura 3.5: Las dos imágenes de la izquierda corresponden a búsquedas usando el método RANSAC de emparejamiento robusto con umbral 5. En ambos casos el programa afirma encontrar el objeto, pero es un resultado falso. Las dos imágenes de la derecha pertenecen a la misma búsqueda, pero con umbral 1. Ahora sólo muestra un falso positivo.

En las pruebas preliminares se ha obtenido que el algoritmo falla al afirmar encontrar el objeto un 59% de las veces. Tras optimizar los parámetros de búsqueda ese porcentaje queda reducido al 37%. Este porcentaje es mucho menor cuando el objeto buscado aparece con nitidez y sin reflejos, pero no parece un comportamiento aceptable para muchas posibles aplicaciones del programa.

## 3.3 Implementación de mejora. Reducción adicional de falsos positivos

Cuando el objeto no se muestra nítido, y el programa no es capaz de extraer suficientes puntos de interés en él dentro de la imagen de búsqueda, si hay un número alto de falsos emparejamientos en el entorno, el resultado de la homografía suele ser incoherente.

Tras aplicar el programa de búsqueda en varios entornos engañosos se deduce de los resultados observados de objeto encontrado que son falsos, el polígono resultante de la homografía no suele ser un cuadrilátero.

Los casos de falso positivo suelen mostrar uno de los siguientes resultados:

- Dos de las esquinas del cuadrilátero resultante de la homografía coinciden, quedando el falso objeto enmarcado en un triángulo.
- Dos segmentos del marco de homografía se cruzan.
- Uno de los ángulos del cuadrilátero es superior a 180º.

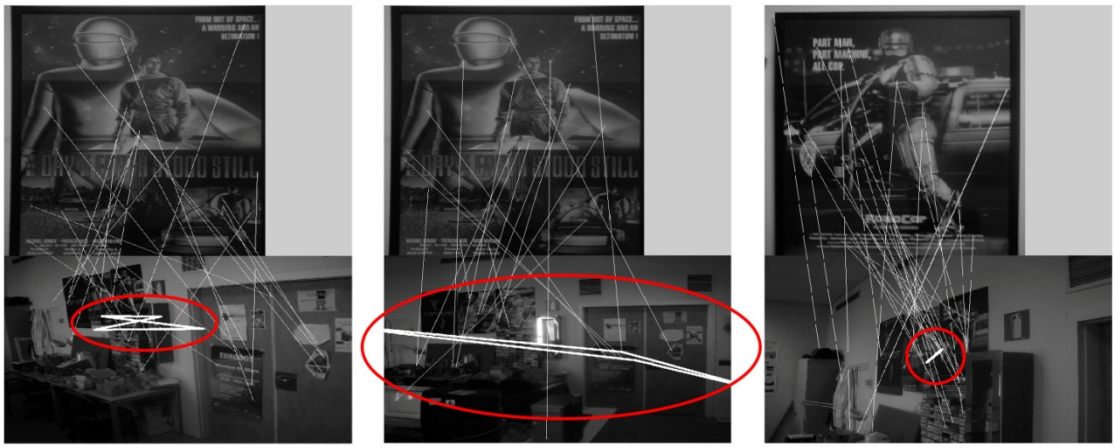


Figura 3.6: Ejemplos de falsos resultados positivos. El cuadrilátero no incluye dentro al objeto buscado ni tiene una forma coherente.

Se decide implementar una función dentro del programa que descarte las soluciones en las que se produzca cualquiera de estos casos.

Esta función se aplica sobre dos lados opuestos del posible cuadrilátero. Lo primero que comprueba es que ninguno de los dos segmentos sea de longitud nula o que compartan uno de los puntos extremos. Si los segmentos pasan esta prueba, se atiende a la posición relativa entre ellos. Se busca la intersección de una recta con la otra y se comprueba si esa intersección ocurre entre segmentos. Si es así se descarta

el resultado positivo. En caso de pasar esta prueba favorablemente se realiza otro paso, en el que con uno de los segmentos forzado sobre el eje X, se observa si los extremos del otro segmento tienen la coordenada Y de distinto signo (figura 3.7). En ese caso también se tomaría el resultado por un falso positivo.

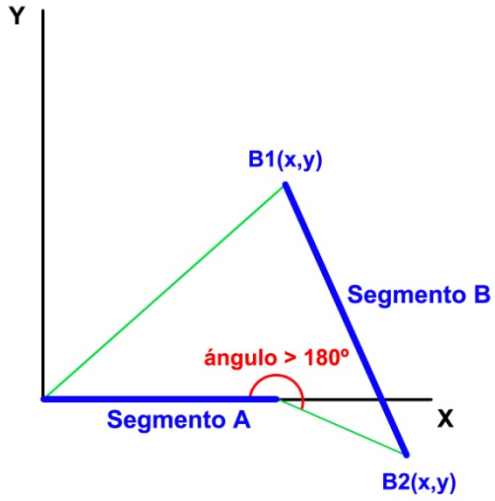



Figura 3.7: Las coordenadas verticales de los puntos que definen el segundo segmento tienen distinto signo, en consecuencia el cuadrilátero formado tiene uno de sus ángulos superior a 180º.

Si los dos lados pasan favorablemente la prueba, se aplica la misma función a los otros dos lados. Si también pasan el test el resultado se da por bueno.

Las búsquedas realizadas, tras la implementar esta nueva función en el programa, eliminan prácticamente todos los falsos positivos en los resultados (figura 3.8). En todas las pruebas tan sólo se ha filtrado un resultado en el que el objeto encontrado podía ser considerado como una solución errónea (y no del todo, pues el objeto estaba dentro del cuadrilátero como se ve en la figura 3.11, aunque este era mucho mayor de lo normal). Aún así, configurando un mejor ajuste de las condiciones de búsqueda, incluso ese resultado desaparece como solución fallida.


Figura 3.8: Resultados al mismo problema planteado en el experimento de la figura 3.3 tras implementar la nueva función. Desaparecen los falsos positivos.

## 3.4 Configuración de la extracción del descriptor

La extracción del descriptor es parametrizable alterando ligeramente el código de la aplicación. Por defecto, se ha configurado como se describe a continuación.

***cvExtractSurf*** es la función que busca los puntos característicos y les asigna un descriptor. Para cada punto la función devuelve su localización, tamaño, orientación, y el descriptor, que puede ser básico o extendido.

Por defecto, la aplicación extrae descriptores SURF de 128 elementos (SURF128), que es la "versión precisa". También se puede configurar como "SURF básico" (SURF 64) y los resultados son muy parecidos. SURF128 es ligeramente más lento, aunque a penas es perceptible en la extracción de características. En el proceso de emparejamiento es donde se puede llegar a percibir ese tiempo extra, ya que se dobla el número de características a comparar.

También es modificable el número de octavas y capas. Por defecto, utiliza 3 octavas de 4 capas cada una.

Por último, dentro de la misma función, también es modificable el umbral Hessiano. Este umbral establece que el determinante de la Hessiana de un punto debe ser mayor a un valor dado para que el punto se tenga en cuenta y se extraiga el descriptor. Por defecto el valor es 500. Los valores entre 300 y 500 forman un buen rango para empezar. Un mejor ajuste requiere conocer las características concretas del objetivo de una aplicación. El valor umbral puede depender de la media local de contraste y de la nitidez de la imagen.

Este umbral también afecta al tiempo de cómputo de la aplicación. Un valor bajo permitirá la aceptación de más puntos de interés que después deben ser incluidos en el proceso de emparejamiento. Un valor alto, mejora el tiempo de cómputo, pero existe el riesgo de eliminar puntos de interés útiles.

## 3.5 Configuraciones de emparejamiento

El programa, adicionalmente, implementa varias configuraciones de búsqueda de emparejamiento de puntos y de objeto. Estas opciones son:

- Elección entre método FLANN o “naive”, para la búsqueda de emparejamientos entre puntos de distintas imágenes mediante métodos de vecino más próximo.
- El método FLANN permite a su vez la configuración de un método elegido entre varias opciones, para la búsqueda del vecino más próximo.
- Elección entre método RANSAC y LMEDS, para conseguir un emparejamiento robusto del objeto.
- El método RANSAC requiere la configuración de un valor de umbral.

### *3.5.1 Resultados con método naive*

El método “naive” (simple), que implementa como opción este programa, da unos resultados muy similares a los que se obtienen mediante FLANN, tal como está configurado por defecto, con el método de búsqueda de vecino más cercano de cuatro

Randomized KD-Trees con búsqueda en un máximo de 64 hojas. En cuanto al tiempo de cálculo, también es muy parecido a los métodos de FLANN, incluso dependiendo de la configuración de este último, el “naive” puede en ocasiones ser más rápido.

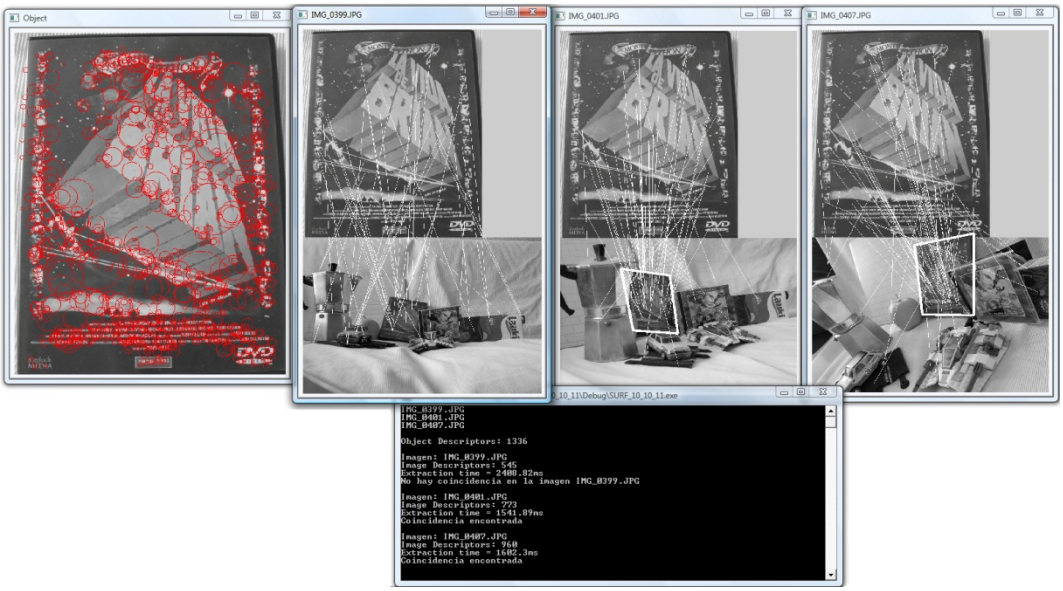



Figura 3.9: Ejemplo de localización de objetos con naive.

Aún así su comportamiento es ligeramente inferior al del método perteneciente a FLANN. En algunas ocasiones, mediante el método simple, el programa no es capaz de encontrar un objeto que, sin embargo, si localizaría con un método FLANN correctamente configurado (figuras 3.9 y 3.10).

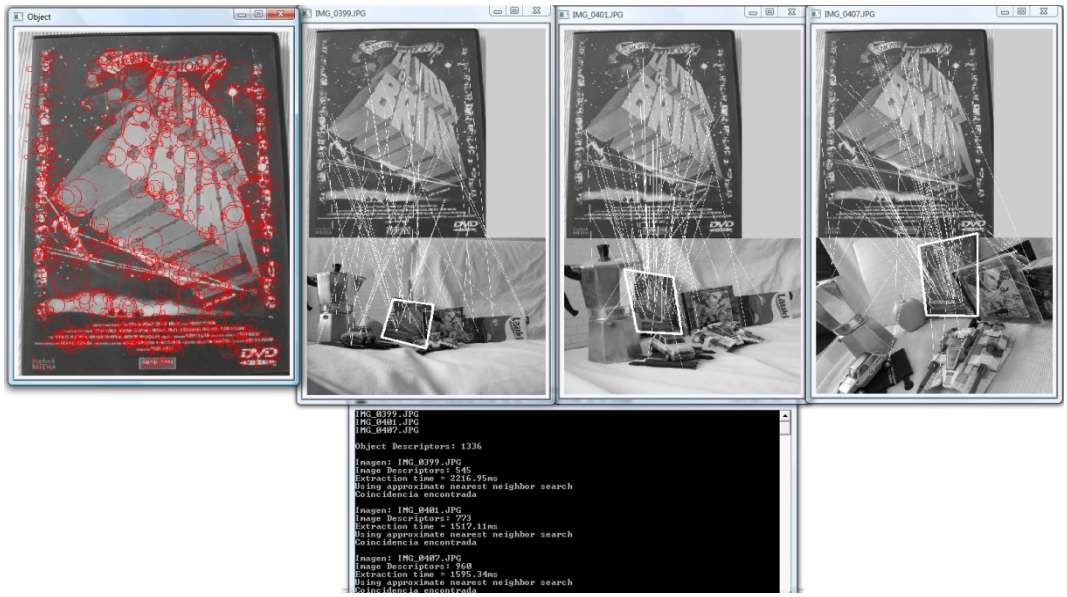



Figura 3.10: Misma búsqueda que en la figura 3.9, pero usando el método de KD-Trees aleatorios con FLANN. En esta ocasión la aplicación es capaz de encontrar el objeto en la primera imagen.

### *3.5.2 Resultados con método FLANN*

FLANN (Fast Library for Approximate Nearest Neighbors) contiene una colección de algoritmos optimizados para la búsqueda del vecino más cercano para grandes conjuntos de datos. De los métodos que proporciona FLANN, en este proyecto se va a estudiar el comportamiento usando el de los KD-Trees aleatorios. Es suficiente para observar cómo afectan los cambios a la solución. Pero, si se desea, es posible modificar esta aplicación para emplear los otros métodos que proporciona la biblioteca.

El método que emplea KD-Trees aleatorios, consiste en la formación uno o más arboles de datos, en los que durante el aprendizaje, cada hoja de cada árbol recibe variables de un conjunto aleatorio de elementos.

Este método permite jugar principalmente con dos variables de ajuste: El número de KD-Trees y el número máximo de hojas visitadas en cada KD-Tree.

En ***KDTreeIndexParams*** se especifica el número de KD-Trees paralelos que se van a emplear (buenos valores de prueba entre 1 y 16).

Tras muchos tests realizados modificando el número de árboles se deduce que en la práctica no hay muchas variaciones en los resultados. Puede haber excepciones, en algunas situaciones críticas, donde hay pocos puntos emparejados correctos frente a los falsos, estos emparejamientos falsos no están muy lejos de donde podrían haber estado los reales, y se utiliza un método robusto de emparejamiento con un umbral muy pequeño. En estos casos con un número muy alto de arboles, es posible localizar el objeto a costa de un mayor tiempo empleado en construir las estructuras de datos y en buscar al vecino más cercano. Cuando se usa tan solo un árbol también es fácil perder el objeto.

Los resultados mostrados a continuación tienen una configuración de búsqueda de KD-Tree alterando el número de árboles, con un 64 como número máximo de consultas para la búsqueda del vecino más cercano, y método RANSAC con umbral 1, en imágenes de tamaño 640x480.


Figura 3.11: Búsquedas alterando el número de árboles kd usados. La imagen de la izquierda es el resultado de la búsqueda con 1 solo árbol. La imagen de la derecha utiliza 4 árboles pero el resultado es muy similar al obtenido con 2, 3 y 5 árboles.

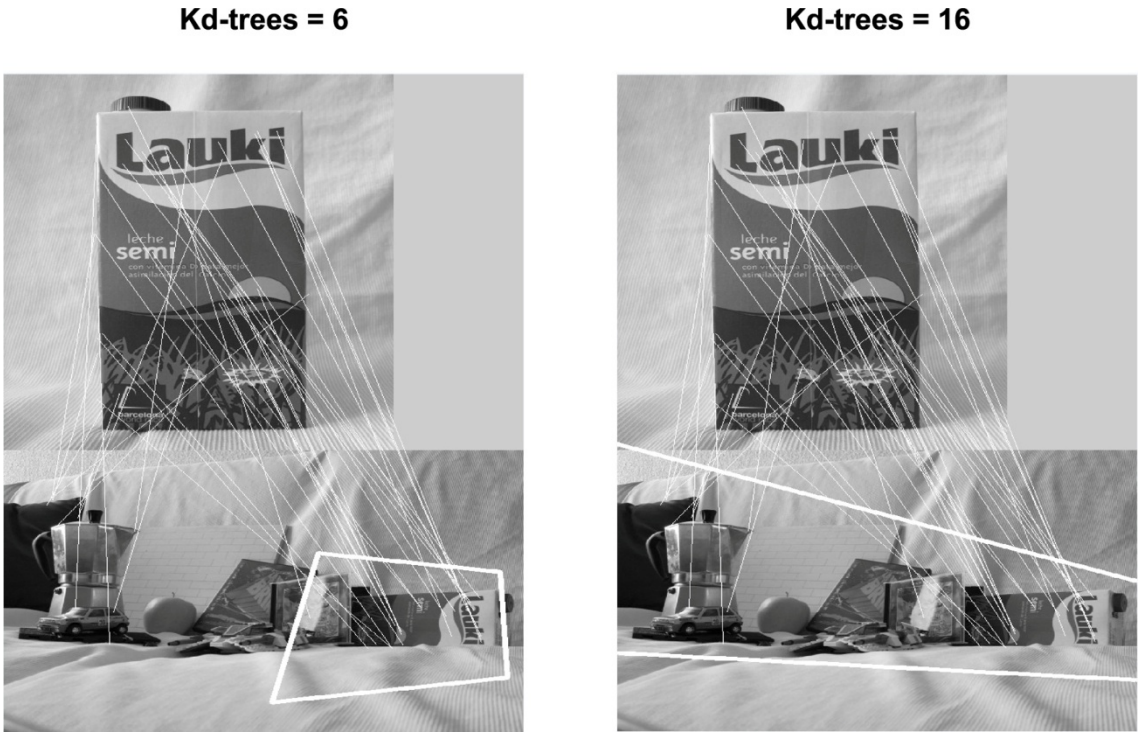



Figura 3.12: Búsquedas alterando el número de árboles kd usados. La imagen de la izquierda es el resultado de la búsqueda con 6 árboles. A partir de 7 arboles el resultado vuelve a empeorar hasta el empleo de unos 40 kd-trees. Con el uso más de 40 árboles la localización es buena a costa de mayor tiempo de cálculo.

Observando las figuras 3.11 y 3.12 se puede deducir que la elección de un número de árboles en lugar de otro, en algunos casos especiales puede dar resultados poco esperados. En esta prueba en particular, el programa tiene problemas para concretar la posición del objeto. Con 1 árbol no lo encuentra. Sólo con 6 árboles, y otra vez a partir del uso de 40 árboles, la localización del objeto es aceptable.

Sin embargo, lo observado en la prueba anterior no significa que un número alto de arboles siempre sea mejor (figura 3.13), ni que un número de 6 árboles sea el óptimo para cualquier caso (figura 3.14).

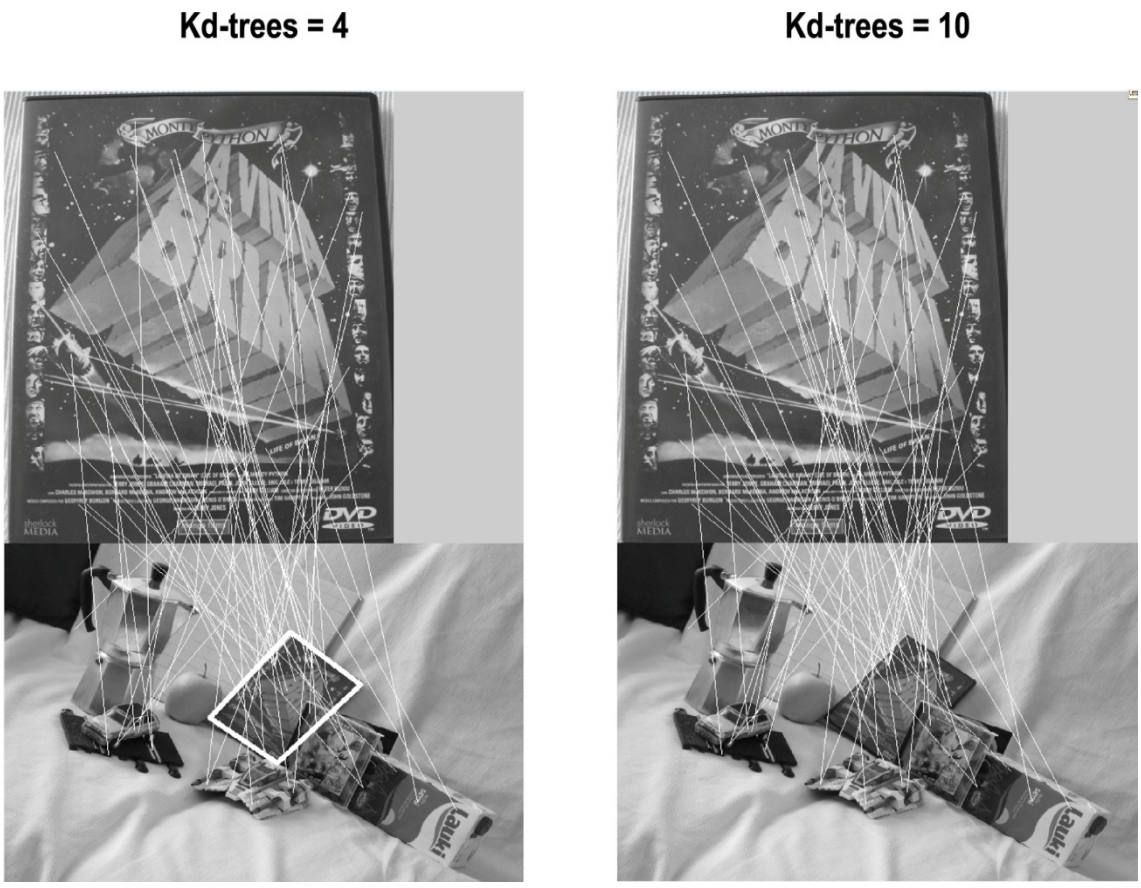



Figura 3.13: Búsquedas alterando el número de árboles kd usados. La imagen de la izquierda es el resultado de la búsqueda con 4 árboles. La imagen de la derecha emplea 10 árboles. El caso es el mismo, pero la segunda vez el programa no encuentra al objeto.

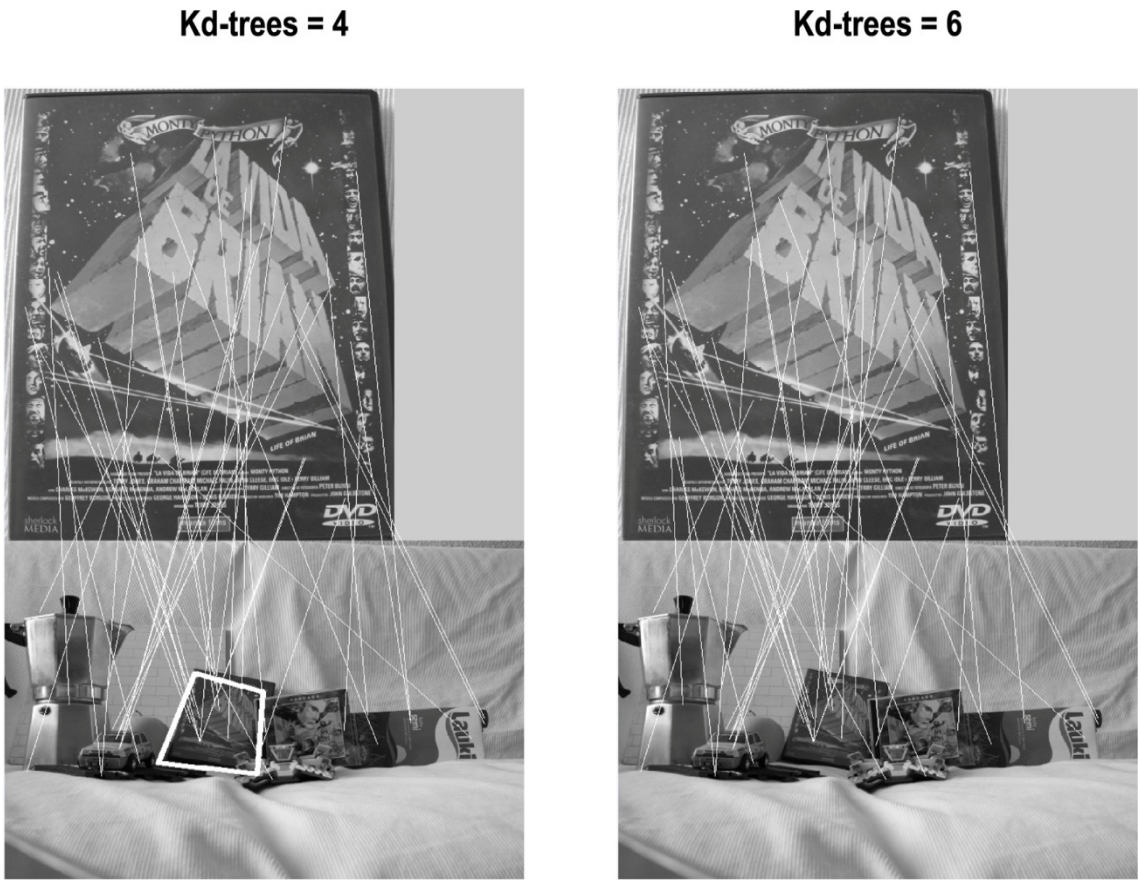



Figura 3.14: Búsquedas alterando el número de árboles kd usados. La imagen de la izquierda es el resultado de la búsqueda con 4 árboles. La imagen de la derecha emplea 6 árboles. En esta ocasión el resultado obtenido con 4 es mejor que con 6.

De todos modos, las situaciones especiales que se acaban de mostrar, no deben ser motivo de preocupación sobre buen el funcionamiento del programa. Más adelante se verá que se soluciona fácilmente jugando con otros parámetros.

Para la mayoría de los casos, en los que el objeto es fácilmente visible, no está tan alejado que apenas se pueden extraer puntos de interés, y donde la nitidez de la imagen es buena, no se observan variaciones en los resultados (figura 3.15) cambiando el número de árboles KD. Un número de cuatro arboles puede ser un buen valor para la mayoría de las ocasiones.

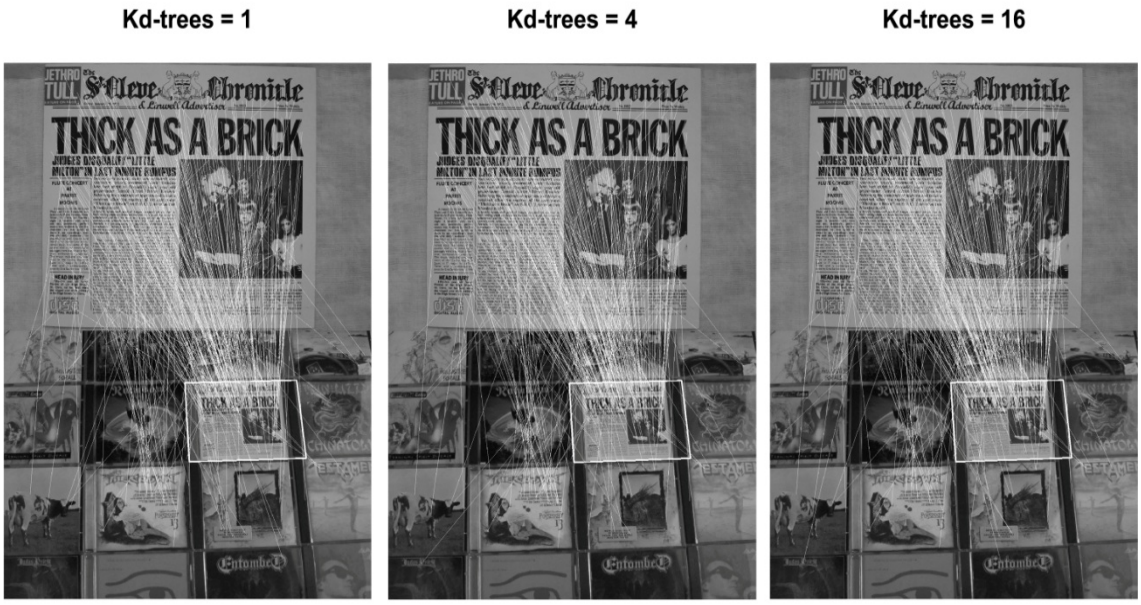
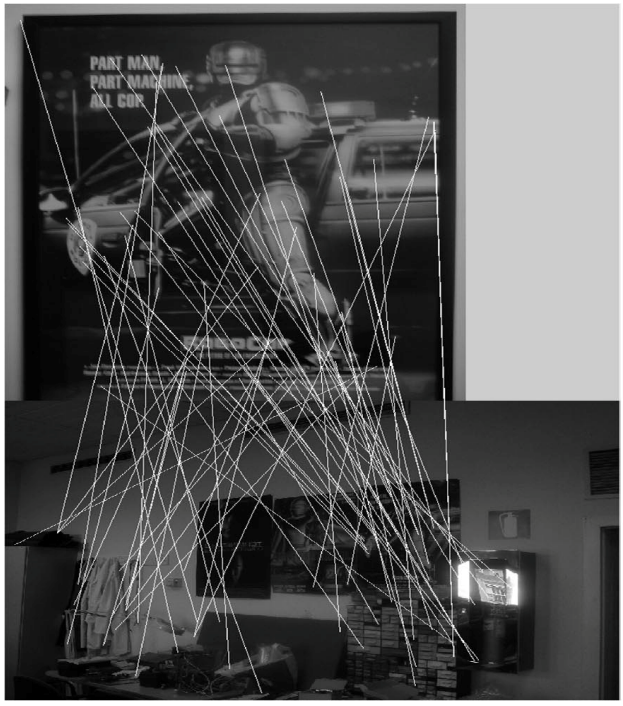



Figura 3.15: Búsquedas alterando el número de árboles kd usados. El resultado es bueno en cualquier caso siempre que las condiciones de la imagen no obstaculicen la extracción de puntos de interés.

Después e fijar el número de árboles se define el uso de la búsqueda paralela del vecino más cercano. En este punto, en ***SearchParams*** se establece el número máximo de hojas que se visitan durante la búsqueda. Valores altos de esta variable mejoran la precisión, pero también incrementan el tiempo de cálculo. Por ello según el tipo de aplicación (características del objeto y del entorno, tamaño imagen, abundancia de puntos de interés…), se debe decidir la opción más conveniente.

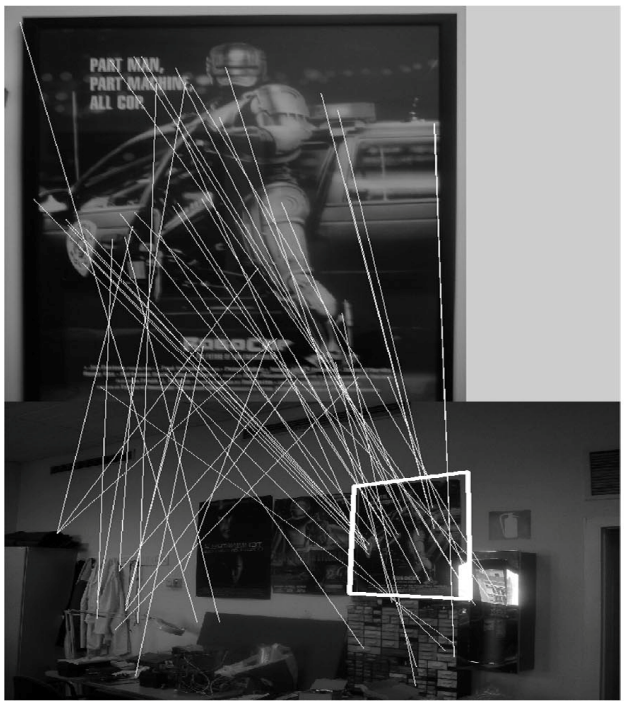
A continuación aparecen ejemplos de búsqueda con diferentes valores para esta variable:




**Kd-Tree = 4**

**Nº Max Consultas = 8**

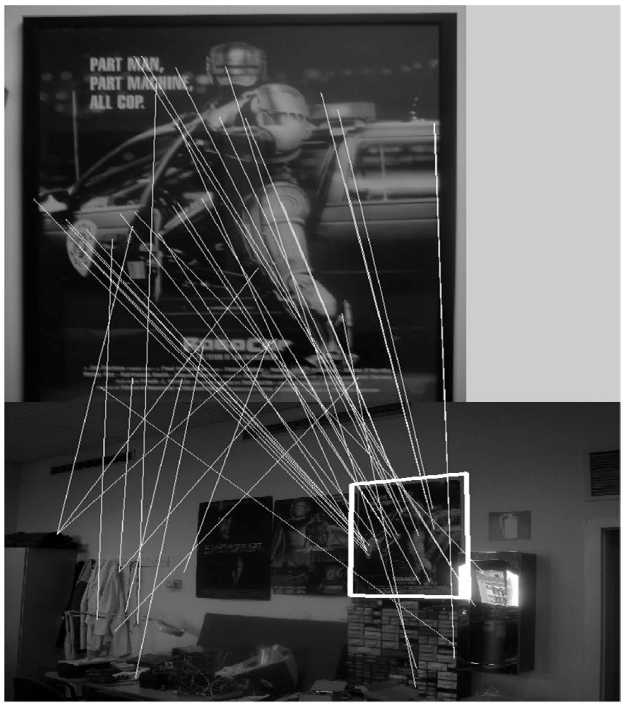
Figura 3.16: Búsqueda de objeto con 4 kd-trees y con número máximo de visitas a hojas 8. El programa no es capaz de eliminar suficientes emparejamientos falsos para encontrar el objeto.




**Kd-Tree = 4**

**Nº Max Consultas = 16**

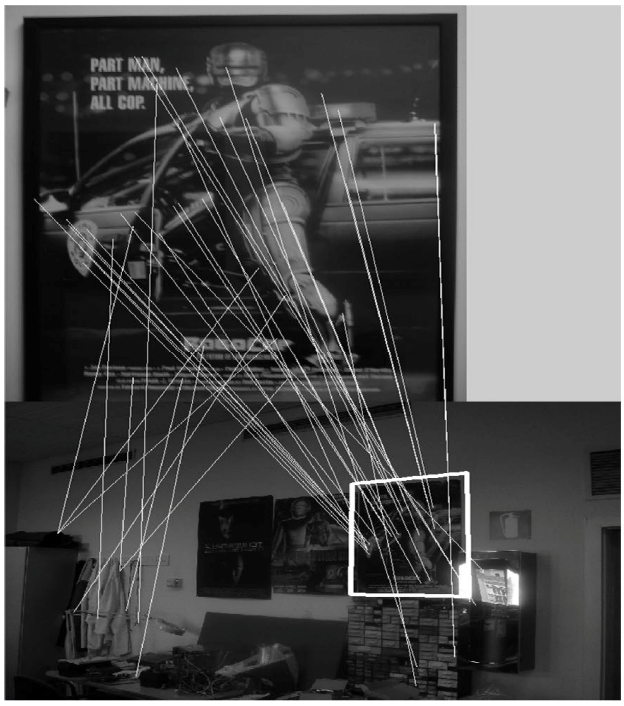
Figura 3.17: Búsqueda de objeto con 4 kd-trees y con número máximo de visitas a hojas 16. Muchos emparejamientos falsos se han eliminado de la nube de datos. Es posible detectar el objeto, pero el porcentaje de falsos emparejamientos sigue siendo elevado.




**Kd-Tree = 4**

**Nº Max Consultas = 32**

Figura 3.18: Búsqueda de objeto con 4 kd-trees y con número máximo de visitas a hojas 32. Es posible apreciar otra reducción de falsos emparejamientos.




**Kd-Tree = 4**

**Nº Max Consultas = 64**

Figura 3.19: Búsqueda de objeto con 4 kd-trees y con número máximo de visitas a hojas 64. Se reduce de nuevo la cantidad de emparejamientos espurios. La precisión del marco de homografía es superior.

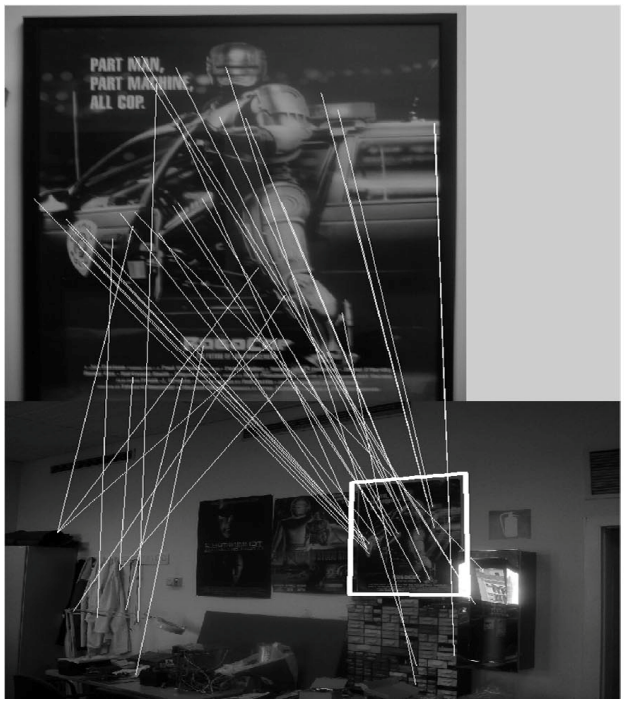



Figura 3.20: Búsqueda de objeto con 4 kd-trees y con número máximo de visitas a hojas 128. El aumento de la precisión apenas es perceptible, y también crece el tiempo de cómputo.

De las pruebas realizadas con valores diferentes de número máximo de visitas a hojas del kd-tree, con diferentes entornos y objetos, se puede concluir que establecer el máximo de visitas a hojas en 64 hojas da suficiente precisión sin incrementar en exceso el tiempo de búsqueda. Valores superiores pueden proporcionar un mejor resultado en la matriz de homografía, pero no es necesario. Valores inferiores, incluso en torno a 20, son suficientes para detectar el objeto, pero quedan sin eliminar muchos puntos emparejados falsos, que pueden estropear la forma de la restricción del objeto.

### *3.5.3 Resultados con método RANSAC de emparejamiento robusto*

La función ***cvFindHomography*** es la que se encarga de encontrar la transformación de perspectiva entre dos planos. Como resultado de su aplicación, se obtiene la matriz de homografía, que trasfiere los puntos una imagen de origen a otra de destino.

Para ello, se aporta la información conseguida en el paso previo. Se deben introducir los puntos (en coordenadas homogéneas) emparejados de la imagen origen, sus correspondientes en la imagen de destino, y seleccionar uno de los métodos de emparejamiento robusto.

También existe la posibilidad de no elegir método, en cuyo caso se usarían todos los puntos emparejados para el cálculo de la matriz de homografía. Pero el resultado no será muy bueno si hay un alto porcentaje de emparejamientos espurios.

En el caso del método RANSAC, es necesario elegir un valor de umbral. ***ransacReprojThreshold*** se considera como el máximo error de reproyección para tratar un par de puntos como puntos de consenso (inlier). Si:

$$\|\mathrm{dstPoints_i} - \mathrm{convertPointHomogeneous}(\mathrm{HsrcPoints_i})\| > ransacReprojThreshold$$

entonces el punto i se considera fuera de consenso (outlier).

La posición de los puntos de origen y de destino, srcPoints y dstPoints, se mide en píxeles. En principio, según el tamaño de la imagen, tiene sentido utilizar valores de umbral entre 1 y 10.

El efecto que tiene en los resultados esta variable, es similar al de cambiar el número máximo de consultas en la búsqueda del vecino más cercano. En este caso, cuanto mayor sea el valor umbral, más emparejamientos falsos afectarán al resultado, y más difícil será la localización del objeto. Y si el valor umbral es demasiado bajo, es posible que se eliminen tantos emparejamientos que impidan restringir su posición en la imagen de destino.

Las siguientes figuras muestran cómo afecta esta variable al resultado. Las imágenes empleadas tienen un tamaño de 600x800. El método de emparejamiento de puntos es el de los KD-Trees aleatorios con 4 árboles y 64 consultas máximo. Para ver el efecto provocado, se elige un valor umbral coherente para el tamaño de la imagen igual a 5, y a partir de ahí se va reduciendo.


Figura 3.21: Búsqueda de objeto con método RANSAC y umbral 5. El programa, debido a los brillos de la imagen de destino, no es capaz de extraer suficientes puntos de interés y no encuentra la imagen.


Figura 3.22: Búsqueda de objeto con método RANSAC y umbral 4. Con la reducción del valor de umbral se localiza el objeto, pero la precisión de la restricción no es muy buena.


Figura 3.23: Búsqueda de objeto con método RANSAC y umbral 3. Una nieva reducción muestra una solución aceptable.


Figura 3.24: Búsqueda de objeto con método RANSAC y umbral 1. El resultado es prácticamente idéntico al obtenido con umbral 2 y 3.

Con los resultados observados, parece que con unos valores de umbral entre 1 y 3 para este tipo de imágenes los resultados son buenos. Con valores umbral mayores a 5 puede haber dificultades en algunos casos en los que la nitidez del objeto es mala. Por lo que parece que cuanto menor sea el valor umbral, mejor será el resultado.

¿Pero qué ocurre si se vuelve a analizar las imágenes que daban problemas en el apartado de configuración del KD-Tree?

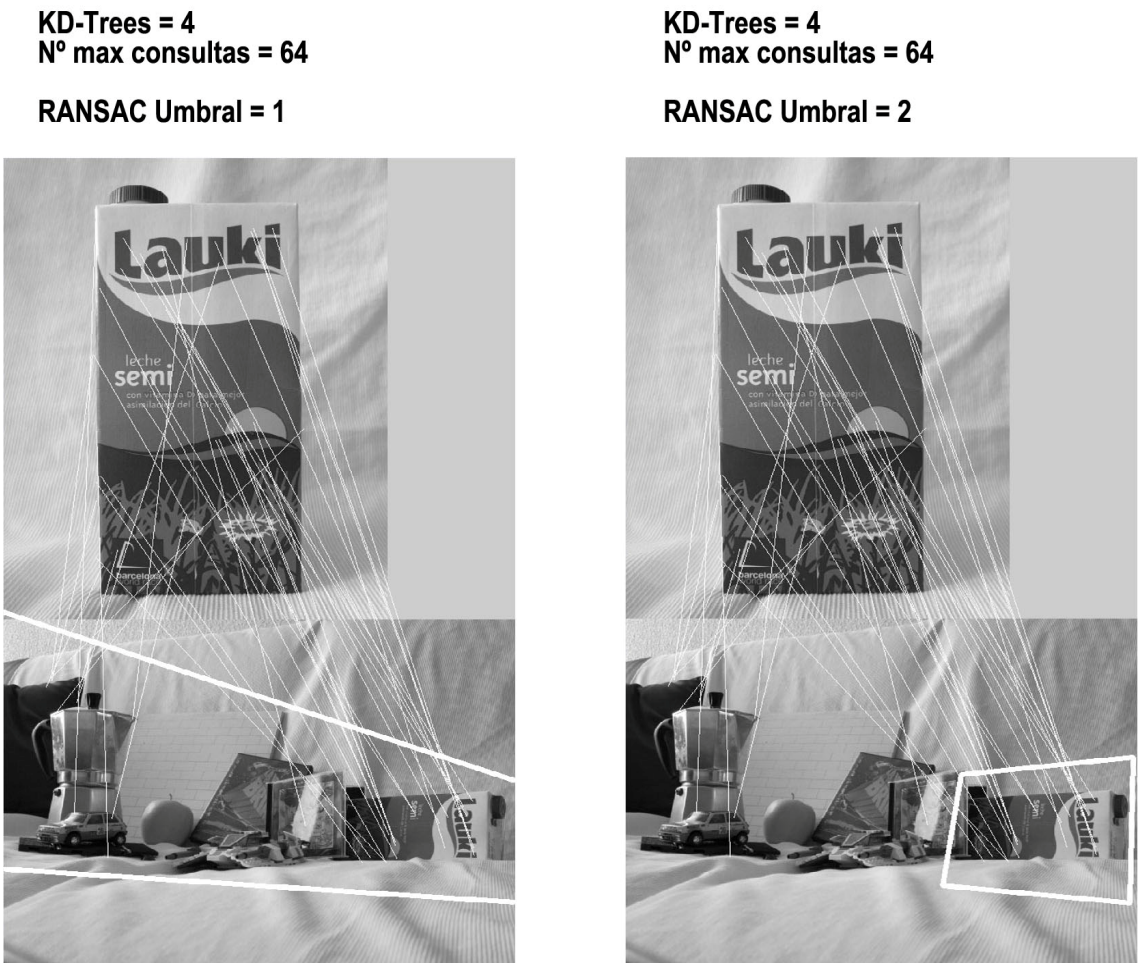



Figura 3.25: Búsqueda de objeto con método RANSAC con umbral 1 (izquierda) y umbral 2 (derecha). Con umbral 1 se obtienen resultados imprevistos al variar el número de KD-TRees. Con un umbral mayor, la localización es buena para cualquier número de KD-Trees mayor a 1.

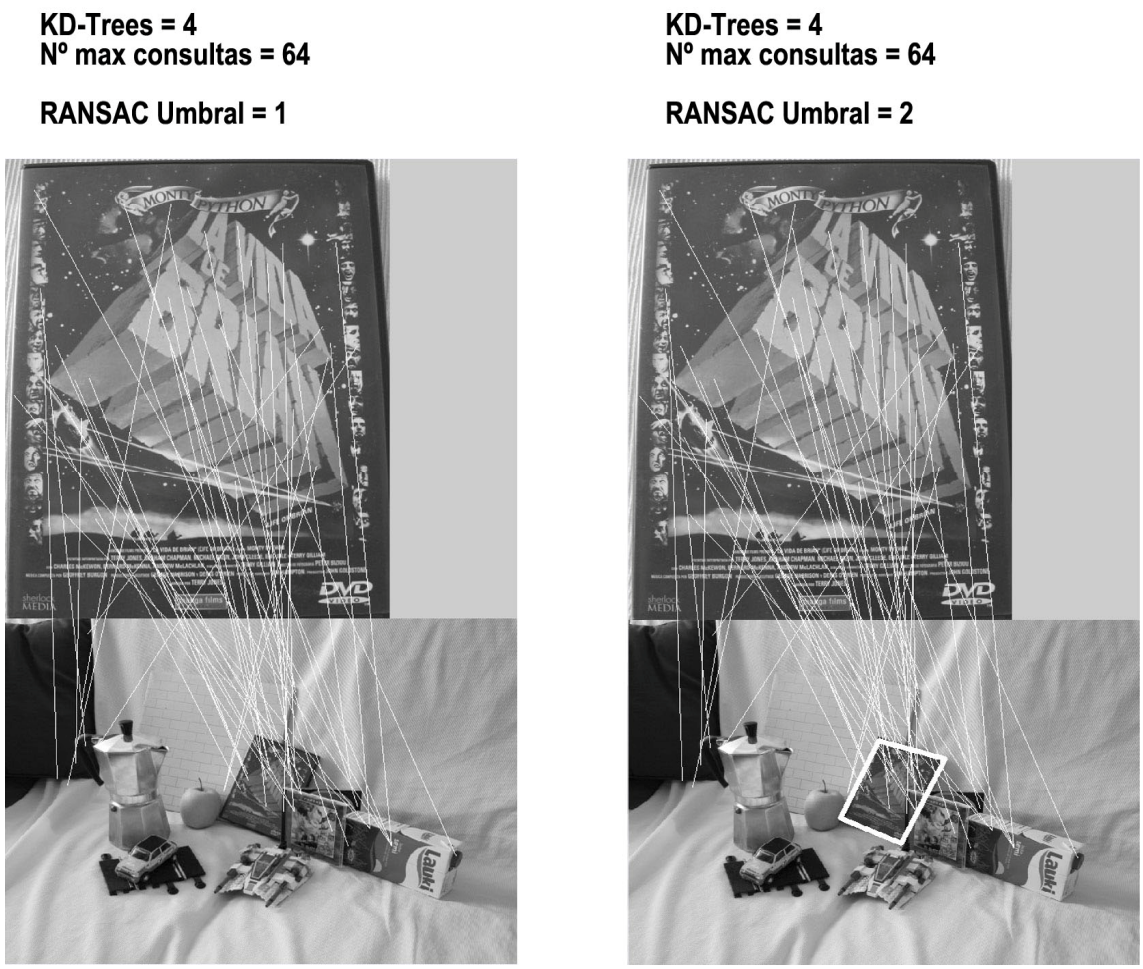



Figura 3.26: Búsqueda de objeto con método RANSAC con umbral 1 (izquierda) y umbral 2 (derecha). En esta ocasión no se podía localizar la imagen en la mayoría de los casos cando el umbral RANSAC era 1, pero tras aumentarlo, deja de ser necesario seguir jugando con los parámetros del KD-Tree para encontrar fácilmente el objeto.

Con los resultados obtenidos en estas pruebas realizadas de nuevo, con un valor mayor para el umbral del método RANSAC, se ve que un valor muy bajo de esta variable también puede ser perjudicial.

Para tamaños de imagen próximos a 800x800, los valores 2 y 3, indistintamente, son los que mejor resultado han dado en las pruebas realizadas.

### *3.5.4 Resultados con método LMEDS de emparejamiento robusto*

En la misma función ***cvFindHomography***, se cambia el método a LMEDS. Para ello, sólo hay que sustituir la variable de método de emparejamiento robusto, el valor CV_RANSAC por CV_LMEDS.

El método RANSAC puede manejar prácticamente cualquier rango de emparejamiento de puntos falsos, pero necesita un valor de umbral para distinguir *inliers* de *outliers*. La ventaja del método LMEDS es que no necesita un valor umbral, pero sólo funciona correctamente cuando hay más de un 50% de *inliers*.


Figura 3.27: Ejemplos de búsqueda con LMEDS. En la imagen de la izquierda la localización es precisa, pues el porcentaje de falsos emparejamientos es pequeño. En la segunda imagen el porcentaje es mayor y la restricción del objeto mucho peor.

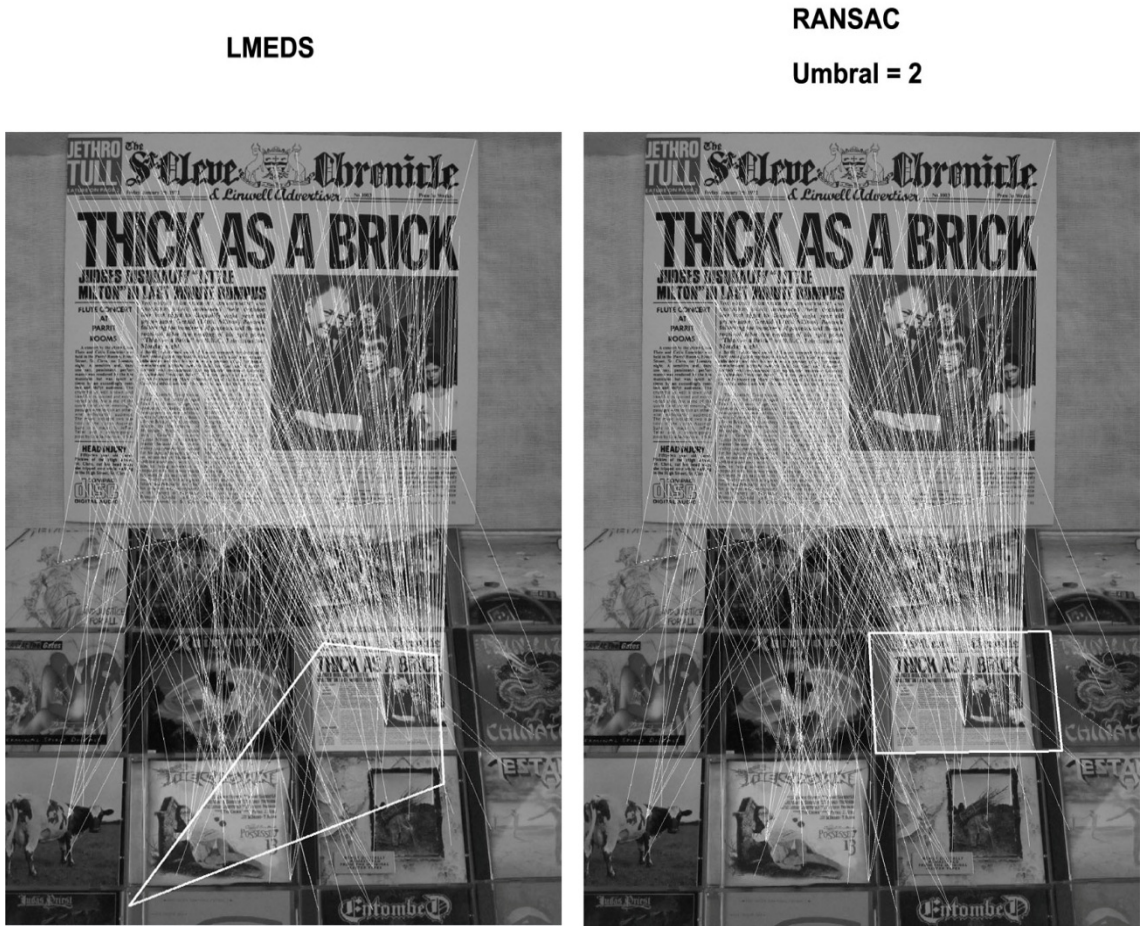



Figura 3.28: Método LMEDS frente a RANSAC. Cuando el porcentaje de emparejamientos espurios puede ser elevado conviene evitar el método LMEDS.

Si la aplicación en la que se va a emplear el programa no incluye situaciones que puedan “despistar” al emparejador, este método resulta una buena opción.

Cuando el número de emparejamientos espurios es elevado, puede perder al objeto en la imagen de destino, o tener poca precisión en el resultado de la homografía

## 3.6 Ejemplos y análisis de resultados en condiciones especiales

Por lo visto hasta ahora, en cuanto a la configuración del programa, parece que una posible configuración óptima para la mayoría de los casos, con imágenes de tamaños comprendidos entre 480x 640 y 1200x1600, es la siguiente:

- Descriptor SURF 128. Umbral Hessiano 500, 3 octavas, 4 capas por octava.
- Método de búsqueda de emparejamiento de puntos FLANN, KD-tree, 4 árboles.
- Nº de consultas máximo en búsqueda de vecino más próximo 64.
- Método de emparejamiento robusto., RANSAC, umbral=2.

Se ha escrito que SURF es un descriptor robusto a transformaciones en la imagen. A continuación, mediante algunos ejemplos, se pondrá a prueba esta afirmación.

### *3.6.1 Rotaciones de objeto sobre el mismo plano de la imagen.*

SURF debe ser robusto a las rotaciones. En la siguiente figura, mediante un ejemplo de un libro colocado en el mismo entorno, con distinto ángulo según un eje aproximadamente perpendicular al plano de la imagen, el programa demuestra no tener problemas para localizar el objeto.

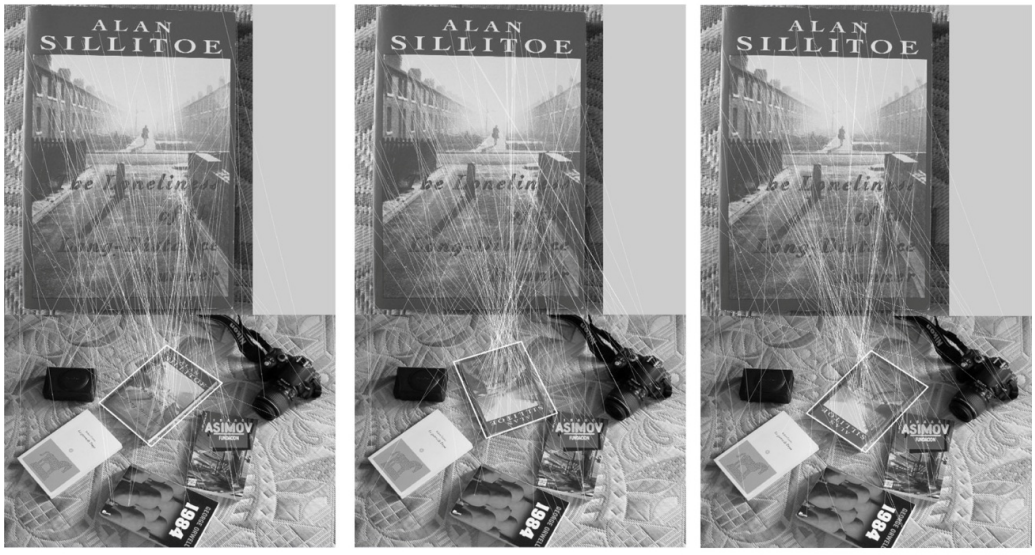



Figura 3.29: En un mismo entorno el objeto buscado está girado sobre sí mismo. Surf demuestra ser invariable frente a rotaciones.

### *3.6.2 Diferentes puntos de vista del mismo objeto.*

En este caso el objeto se ha fotografiado desde varios ángulos. El descriptor, se comporta bien ante cambios de perspectiva forzados.

No hay problemas mientras el cambio de ángulo no sea tan grande que el algoritmo no sea capaz de extraer suficiente información del objeto en la imagen de destino.

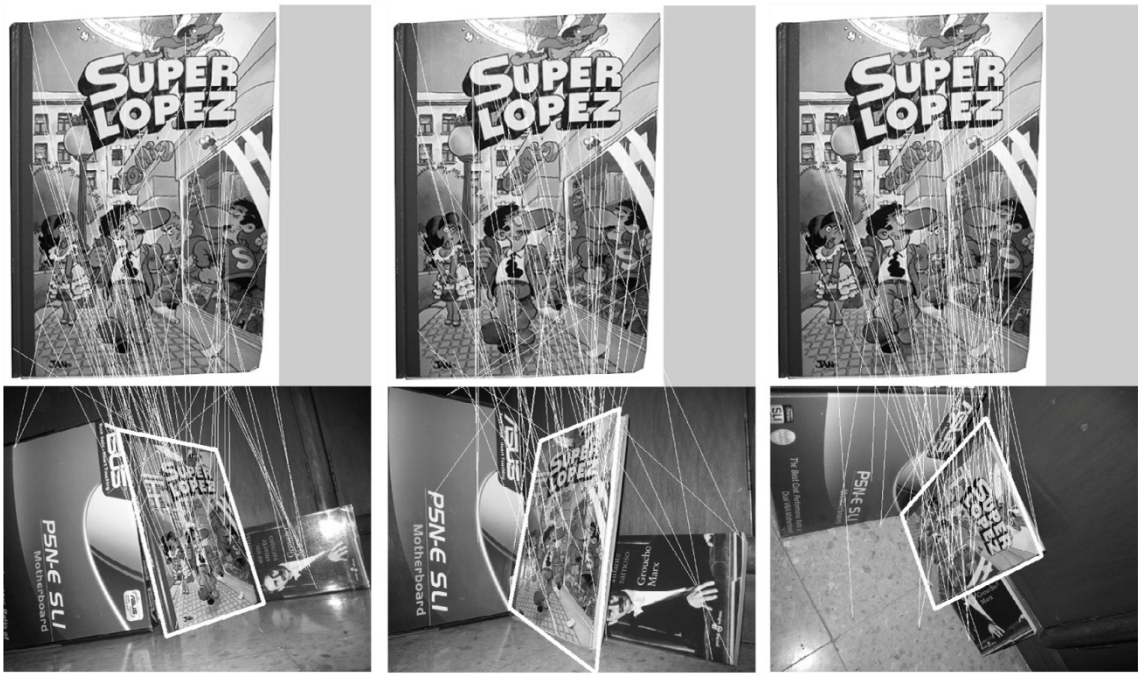



Figura 3.30: Búsquedas del mismo objeto con fotografías tomadas desde distintos ángulos.

Mientras que en rotaciones sobre un eje aproximadamente perpendicular al plano de la imagen, si el detector funciona en un caso, casi se puede asegurar que funcionará siempre, en el tipo de giros de este ejemplo hay más factores que pueden afectar al resultado. La cantidad y el tipo de información que el descriptor es capaz de extraer del objeto, puede ser determinante en un entorno dificil, o cuando se busca un objeto difícil.

Un entorno dificil, sería donde el algoritmo encuentra muchos puntos de interés falsos en el entorno.

Un ejemplo de objeto difícil, puede ser, cuando este presenta un patrón que se repite, es decir, tiene muchas zonas con información similares dentro de él.

A veces, es conveniente limitar el tamaño de la imagen para reducir los detalles, y obligar al extractor de puntos de interés a centrarse en los detalles más grandes y característicos del objeto.

### *3.6.3 Cambios bruscos de tamaño.*

SURF es robusto al cambio de escala ¿Pero qué pasa en casos extremos?

En las pruebas realizadas, se ha comprobado que el programa puede detectar un objeto, aunque este sea mayor en la imagen de destino que en la propia imagen del objeto. La restricción resultante por homografía mantiene la precisión, aunque parte del objeto esté fuera de imagen.

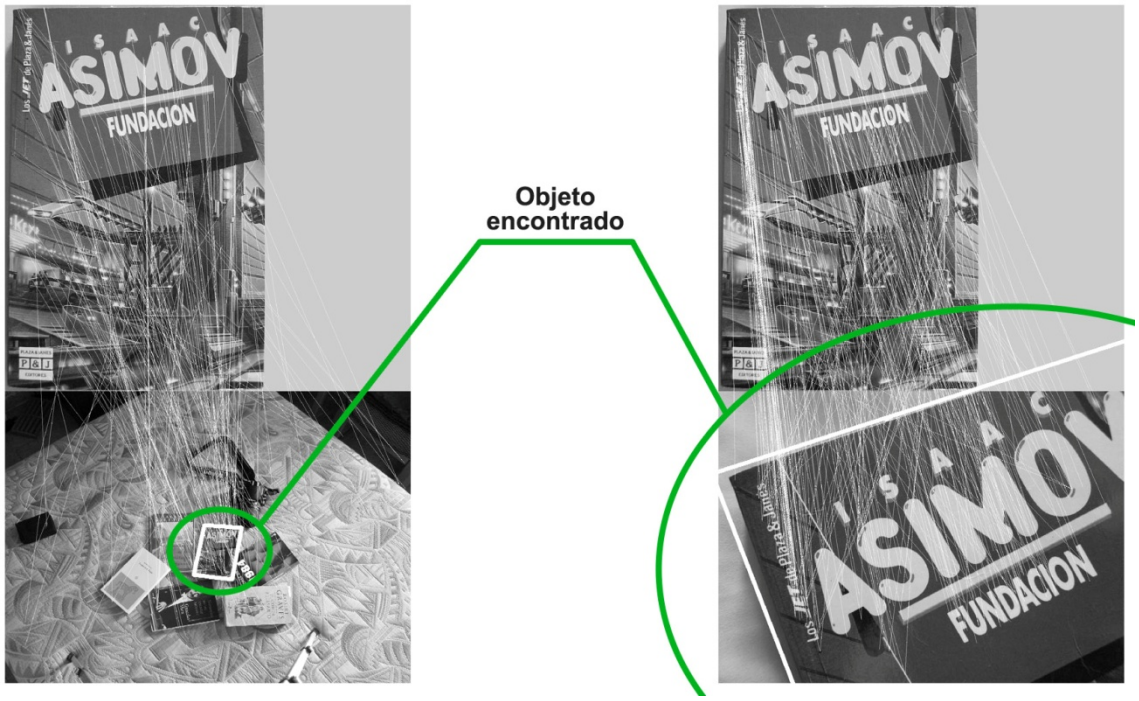



Figura 3.31: En un mismo entorno el objeto buscado está girado sobre sí mismo. Surf demuestra ser invariable frente a rotaciones.

Cuando el objeto buscado está muy alejado, el comportamiento del programa será bueno, siempre que la calidad y el tamaño de la imagen permita extraer puntos de interés del objeto en la imagen de destino. Por ejemplo, en la imagen de la izquierda

de la figura 3.31, el detector funciona si la fotografía tiene un tamaño de 1200x1600, pero falla al usar la misma imagen tomada con una resolución de 480x 640.

### *3.6.4 Objetos parcialmente ocultos*

Mientras sea posible la extracción de una cantidad mínima de puntos de interés del objeto en la imagen de destino, el programa será capaz de, incluso dando un resultado preciso en la homografía.

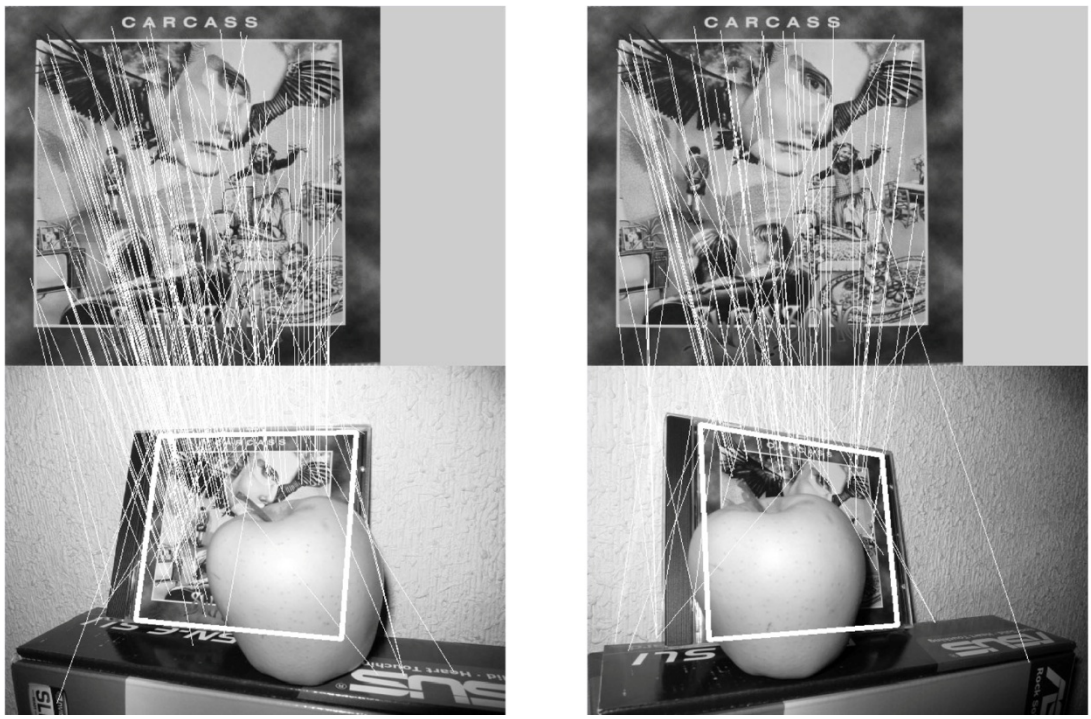



Figura 3.32: Ejemplo de detección de un objeto parcialmente oculto, cubriendo diferentes zonas.

### *3.6.5 Comportamiento del detector frente a objetos no planos.*

Hasta ahora, prácticamente en todas las pruebas, se han buscado objetos planos. Es lógico, pues lo que nos proporciona la cámara es una imagen en dos dimensiones. Pero el detector también es útil para detectar objetos con volumen, aunque en estos casos, la restricción no será tan precisa si lo que se pretende es extraer de ella información sobre los cambios de ángulo y posición.

Como siempre, el detector busca los puntos de interés de la imagen de origen en la de destino, por eso cuanto mayor y mejor sea la información aportada en la primera, mejores resultados se obtendrán frente a variaciones en la segunda.

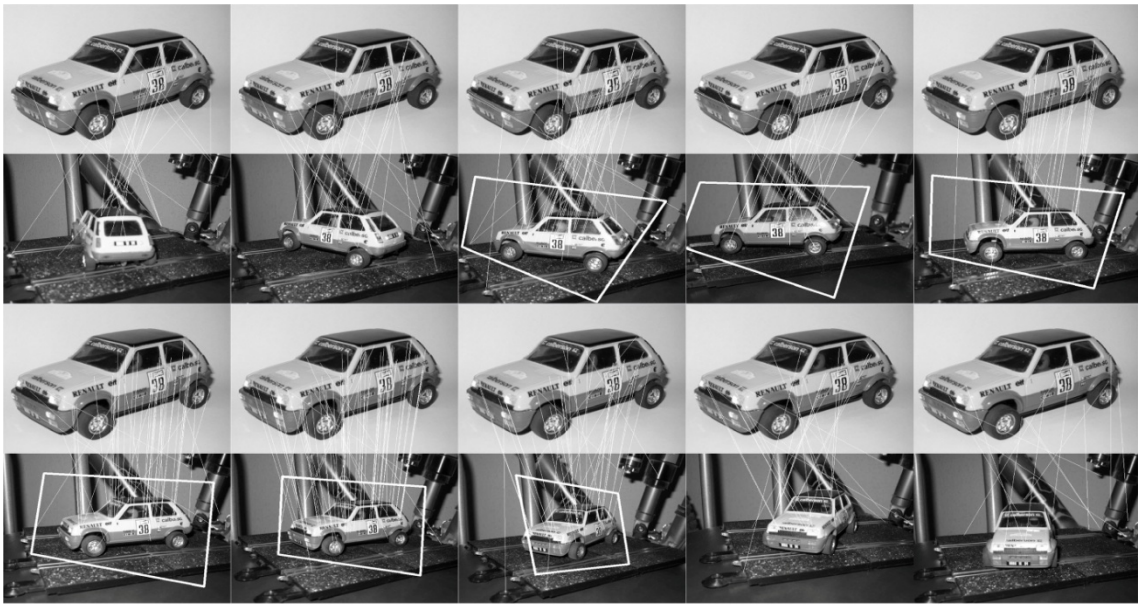


Figura 3.33: Detección de un objeto 3D, visto desde distintos ángulos.

### *3.6.6 Como afecta el tamaño de la imagen*

Cuanto mayor sea la imagen, más detalle se puede percibir en ella, y más descriptores se pueden extraer.

Pero más resolución de imagen no equivale siempre a mejores resultados. Cada tipo de aplicación en la que se puede usar este detector, tendrá unos requisitos que hay que tener en cuenta para elegir la mejor opción.

Una aplicación que debe detectar un objeto en un entorno grande en comparación a este, puede requerir más resolución, para poder extraer suficiente información en la imagen de destino.

Pero cuando este no es el caso, una resolución alta puede dar demasiada información innecesaria sobre el objeto. Cuanto más puntos de interés haya en el objeto, más posibilidades hay de que el programa confunda puntos de la imagen de origen con otros de la de destino, que pertenezcan al entorno. Por eso a veces conviene reducir la resolución, perdiendo detalles que provocan confusión en lugar de ayudar.

Otro factor determinante, para decidir sobre el tamaño de imagen es el tiempo, y la posible cantidad de puntos de interés. A mayor número de puntos de interés, mayor tiempo de computo.

Aunque la cantidad de descriptores depende, además del tamaño de la imagen, de las características del objeto, para tener una idea, se pueden estimar los siguientes valores medios, a partir de las pruebas realizadas:

Imágenes 480x640:

- Descriptores detectados en imagen: entre 400 y 800
- Tiempo medio de extracción de descriptores: 1,2 segundos
- Tiempo comparación y muestra de resultado: 1 segundo

Imágenes 800x600:

- Descriptores detectados en imagen: entre 500 y 900
- Tiempo medio de extracción de descriptores: 1,6 segundos
- Tiempo comparación y muestra de resultado: 1,2 segundos

## III.RESULTADOS

Imágenes 1600x1200:

- Descriptores detectados en imagen: entre 900 y 4000
- Tiempo medio de extracción de descriptores: 7 segundos
- Tiempo comparación y muestra de resultado: 2,5 segundos

# IV. CONCLUSIONES

## IV.CONCLUSIONES

# IV CONCLUSIONES

## 4.1 Conclusiones

Este proyecto ha permitido comprobar el funcionamiento del descriptor SURF (Speeded Up Robust Features) propuesto por Herber Bay, Andreas Ess, Tinne Tuytelaars y Luc Van Gool.

Para ello se ha desarrollado un programa que además de implementar a SURF, incluye una función emparejadora, y otra que recopila las imágenes de una base de datos, para ir ejecutando el detector en cada una de ellas.

En resumen, el programa busca y localiza una imagen objeto dentro de una base de datos de imágenes en las que ese objeto puede aparecer en distintas posiciones y ángulos.

El detector y descriptor SURF ha demostrado ser repetitivo y robusto frente a cambios de escala, rotaciones, e incluso a cambios en la iluminación.

Tanto el algoritmo detector, como el emparejador permiten un amplio rango de configuraciones, que permiten optimizarlo y hacerlo adaptable a gran cantidad de aplicaciones, en las que sea necesario un reconocimiento de objetos.

Además se ha podido comprobar la rapidez de SURF, pudiendo obtener respuestas para algunas aplicaciones en tiempos menores a un segundo, dependiendo de la configuración y del tamaño de las imágenes. Hay que tener en cuenta que en este proyecto se ha empleado por defecto la variante de SURF de 128 elementos, al no ser el tiempo un factor crítico. Pero cambiando una sola variable de la función de extracción se puede trabajar con SURF 64, sin experimentar a penas diferencias en los resultados, pero si reduciendo el tiempo de computo.

## IV.CONCLUSIONES

# V. PRESUPUESTO

## V.PRESUPUESTO

# V PRESUPUESTO

## MATERIAL

| Denominación | Precio Ud. | Cantidad | Total |
|---|---|---|---|
| Visual Studio 2008 Professional | 950 € | 1 | 950 € |

**Total Material 950 €**

## MANO DE OBRA

| Denominación | Precio Ud. | Horas | Total |
|---|---|---|---|
| Desarrollo Software | 30 €/h | 60 | 1800 € |
| Documentación | 30 €/h | 30 | 900 € |

**Total M.O. 2700 €**

**COSTE TOTAL 3650 €**

## V.PRESUPUESTO

# VI. BIBLIOGRAFÍA

## VI.BIBLIOGRAFÍA

# VI BIBLIOGRAFÍA


- Arturo de la Escalera. “Visión por Computador. Fundamentos y Métodos”. Pearson Educación 2001.
- Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool. “Speeded-Up Robust Features (SURF)”. 2008.
- Herbert Bay, Tinne Tuytelaars, Luc Van Gool. “SURF: Speeded-Up Robust Features”, 2006.
- Gary Bradski & Adrian Kaehler. “Learning OpenCV. Computer Vision with the OpenCV Library” O’Reilly 2008.
- Marius Muja & David Lowe. “FLANN – Fast Library for Approximate Nearest Neighbors. User Manual” 2011.
- Paul Viola & Michael Jones.”Rapid Object Detection using a Boosted Cascade of Simple Features”. 2001.
- David G. Lowe. “Object Recognition from Local Scale-Invariant Features”. 1999.

## VI.BIBLIOGRAFÍA

# ANEXO

**ANEXO**

# ANEXO

## OPENCV

Para el desarrollo del programa de detección de objetos, se ha empleado la biblioteca libre OpenCV (Open Source Computer Vision Library).

Esta biblioteca, desarrollada por Intel para el tratamiento de imágenes y aplicaciones de visión por computador, está programada en C y C++, y se puede descargar de http://sourceforge.net/projects/opencvlibrary/

En este anexo, se pretende exponer algunas de las principales funciones y tipos de datos de OpenCV usados en el programa.

### TIPOS DE DATOS BÁSICOS:

#### CvPoint

CvPoint define las coordenadas en un punto en enteros.

#### CvPoint2D32f

CvPoint2D32f define las coordenadas en un punto en coma flotante.

#### CvScalar

La estructura CvScalar es un vector de 4 elementos que resulta muy útil a la hora de acceder a los pixeles de una imagen.

#### CvSize

Tamaño de un rectángulo expresado en Pixels.

#### CvArr

Arbitrary Array. Es un metatipo (tipo de datos ficticio), que se usa sólo como parámetro para especificar que la función acepta arrays de múltiples tipos.

**ANEXO**

## CvMat

Estructura tipo matriz. El concepto es algo más abstracto que el de la matriz de algebra lineal. Los elementos de la matriz no tienen porqué ser simples números. Pueden ser otro tipo que de datos que se desee definir.

## IplImage

IplImage es la estructura básica utilizada para codificar lo que generalmente se llama imagen.

## CvSeq

La estructura de datos (secuencia) CvSeq es la base para todas las estructuras dinámicas de datos de OpenCV.

**PRINCIPALES FUNCIONES EMPLEADAS:**

## cvNamedWindow

int cvNamedWindow(const char* name, int flags)

Crea una ventana que puede ser usada para mostrar imágenes.

## cvDestroyWindow

void cvDestroyWindow(const char* name)

Elimina la ventana de nombre dado.

## cvLoadImage

IplImage* cvLoadImage(const char* filename, int iscolor)

Carga una imagen de un archivo como un IplImage.

**ANEXO**

**cvShowImage**

void cvShowImage(const char* name, const CvArr* image)

Muestra la imagen (segundo parámetro) en la ventana especificada (primer parámetro)

**cvGetSeqElem**

char* cvGetSeqElem(const CvSeq* seq, int index)

Devuelve un puntero a una secuencia de elementos según su índice.

**cvExtractSURF**

void cvExtractSURF(const CvArr* image, const CvArr* mask, CvSeq** keypoints, CvSeq** descriptors, CvMemStorage* storage, CvSURFParams params)

Extrae los descriptores SURF de una imagen. Para cada punto devuelve la localización, orientación y el descriptor.

El resultado es la estructura de datos **CvSURFPoint**.

```
typedef struct CvSURFPoint
{
   CvPoint2D32f pt; // position of the feature within the image
   int laplacian;    // -1, 0 or +1. sign of the laplacian at the
point.
                    // can be used to speedup feature comparison
                    //  (normally features with laplacians of
different
            // signs can not match)
   int size;        // size of the feature
   float dir;       // orientation of the feature: 0..360 degrees
   float hessian;   // value of the hessian (can be used to
            // approximately estimate the feature strengths;
                    // see also params.hessianThreshold)
}
CvSURFPoint;
```

La estructura de los parámetros de la extracción es del siguiente tipo:

```
typedef struct CvSURFParams
{
   int extended; // 0 means basic descriptors (64 elements each),
                 //  1  means  extended  descriptors  (128  elements
each)
```

```
   double hessianThreshold; // only features with keypoint.hessian
         // larger than that are extracted.
                // good default value is ~300-500 (can depend on
the
         // average local contrast and sharpness of the image).
                // user can further filter out some features based
on
         // their hessian values and other characteristics.
   int  nOctaves;  //  the  number  of  octaves  to  be  used  for
extraction.
                //  With  each  next  octave  the  feature  size  is
doubled
         // (3 by default)
   int nOctaveLayers; // The number of layers within each octave
         // (4 by default)
}
CvSURFParams;

CvSURFParams cvSURFParams(double hessianThreshold, int extended=0);
         // returns default parameters
```

**cvFindHomography**

void cvFindHomography(const CvMat* srcPoints, const CvMat* dstPoints, CvMat* H int method, double ransacReprojThreshold, CvMat* status)

Encuentra la transformación de la perspectiva entre dos planos.

**srcPoints:** Coordenadas de los puntos en el plano original.

**dstPoints:** Coordenadas de los puntos en el plano de destino.

**H:** El resultado del cálculo es la matriz de homografía H (3x3)

**Method:** CV_RANSAC o CV_LMEDS

**ransacReprojThreshold:** Umbral del método RANSAC